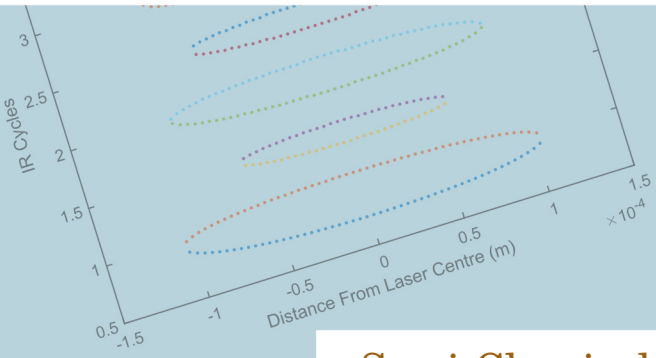
LUND UNIVERSITY



$$U_p = \frac{e^2 E_0^2}{4 m_e \omega^2}$$

Semi-Classical Calculations of Multiple Trajectories in High-Harmonic Generation

Tom Causer

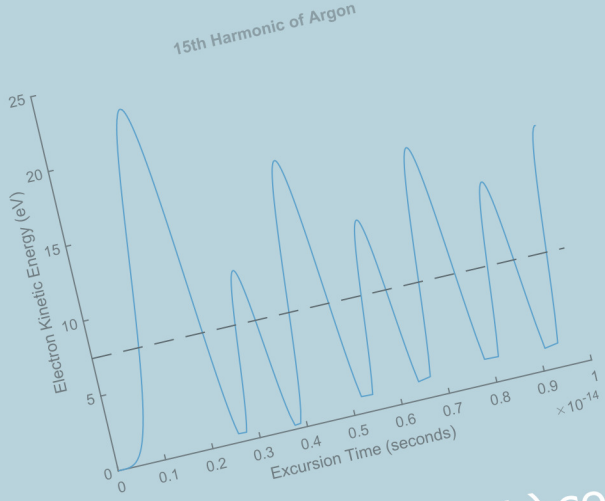
Thesis submitted for the degree: Bachelor of Science
Project Duration: 2 months

Supervised by: Johan Mauritsson
Cosupervised by: Samuel Bengtsson

$$\ddot{x} = -\frac{eE_0}{m_e}\sin(\omega t)$$

$$\dot{x} = \frac{eE_0}{m_e \omega}\left(\cos(\omega t) - \cos(\omega t_i)\right)$$

$$x = \frac{eE_0}{m_e \omega^2}\left(\sin(\omega t) - \sin(\omega t_i) - \omega(t - t_i)\cos(\omega t_i)\right)$$



Department of Physics
Atomic Division
May 2021

Abstract

The purpose of this thesis is to investigate the High-Harmonic Generation (HHG) process by simulating electron trajectories, and their resulting Extreme Ultra Violet (XUV) output. The HHG process is one in which a focused beam of infrared (IR) light can rival (in terms of intensity) the potentials of atoms, which makes tunnelling likely – this sets off a chain of events resulting in an XUV pulse.

The thesis utilises a semi-classical method in order to predict where one might look for further trajectories. This method is programmed into MATLAB in order to run simulations and investigate the results.

The code allows for a deep-dive into different aspects of the process, including laser profile shape, laser properties (wavelength and intensity) and number of atoms within the beam.

One of the conclusions of the thesis is the reaffirmation that finding the third trajectory is not simply a case of knowing what settings to apply, but moreso the principle that trajectories past the second are much less likely to be taken by electrons (even in a purely classical regime, before taking into account phenomena like Electron Wave Packet (EWP) spreading). The thesis also demonstrates an approximation of the HHG process to the tenth trajectory, where usually only up to the second is shown, which provides an interesting insight into how they are grouped around certain harmonics.

Popular Description

Imagine going to the park with your dog and playing fetch. We're going to assume that your dog always at least attempts to fetch the ball once you throw it. Let's also assume that upon returning the ball to you, the dog shows her excitement by barking in a frequency (your dog is a tad odd) that depends on the speed she returns to you. This basic premise takes place on an atomic level within the field of atomic physics, in a process called "Higher-order Harmonic Generation" (HHG).

How hard would you have to throw the ball before it leaves the park and your friend can no longer retrieve it? Or if you have a penchant for annoying others, you might ask the question "How hard do I have to throw this ball so that my dog sounds like a nail scratching a chalkboard?"

These questions have analogues in HHG. The frequency of your dog's bark in this analogy corresponds to the frequency (and by extension: energy) of the light that comes out of the HHG process.

There's more to the process than just you and the dog though – one of the reasons the HHG process can take place at all is because of the ability of electrons to "tunnel through the potential barrier". In our dog analogy, this means that your willingness to play fetch as the owner will change depending on your environment. You are perhaps not so willing to play fetch in Auntie Gertrude's antique store, but once you reach an open field – you almost have to resist throwing the ball.

There are equations that allow for the simulation of this atomic scenario and also for much more interesting questions to be asked. You might think that the way to ensure your loyal companion sounds like she's imbibed helium is simply to throw it as hard as you can. This would be true if she were incapable of tiring. In reality, if you throw it too hard (but still inside the park) then your companion might return to you at a walking pace, sounding like a deflating balloon.

One of the unanswered (or rather unmeasured) questions in the field that studies this, is based upon the premise that your dog physically "misses you" (think of it as running in-between your legs by accident) when she tries to return the ball to you. Upon passing you, she'll turn and come back – and when she does successfully return the ball to you, her speed will still be measurable, and it is guaranteed to be a speed that could have been measured had she not missed you the first time. The problem is – such an event has not yet been experimentally observed; the dog always seems to be successful in returning the ball the first pass despite the theory pointing out that the miss scenario should be possible.

The subject of my thesis is to simulate your visit to the park with the dog and investigate the ways in which we might be able to make your dog run past you instead of returning the ball directly to you.

Abbreviations, Acronyms & Terms

as (attosecond) - 10^{-18} seconds, a unit of time

DFT - Discrete Fourier Transform

FFT - Fast Fourier Transform

EWP - Electron Wave Packet

HHG - High-order Harmonic Generation

IR - Infrared

V/m - Volts per Metre, equivalent to Newtons per Coulomb, a unit of electric field strength.

SFA - Strong Field Approximation

TDSE - Time Dependent Schrödinger Equation

Contents

1 Introduction, Background & Theory

Light is composed of oscillating electric and magnetic fields. In this thesis, the magnetic field component will be neglected – to be specific, there will be the assumption that all light is linearly polarised. The electric fields are taken into account though and need to be considered. A simple way of visualising the oscillations of the electric field is to use a sine function, as in equation 1.

$$\vec{E}(t) \quad = \quad \vec{E}_0 \sin(\omega t) \tag{1}$$

where $\vec{E}(t)$ is the electric field strength [V/m] at time t, $\vec{E}_0$ is the peak electric field strength, ω is angular frequency [rads/s] and t is time [s].

In a linear system, waves obey the superposition principle and can either interact constructively, or destructively. This is the basis for creating *pulses* of light. If a sufficient number of light waves are summed, then it is possible to ensure that there is destructive interference everywhere except at a point of interest. Using this principle, it is possible to create short pulses– this is demonstrated in figure 1. For simplicity, there are only five waves summed.

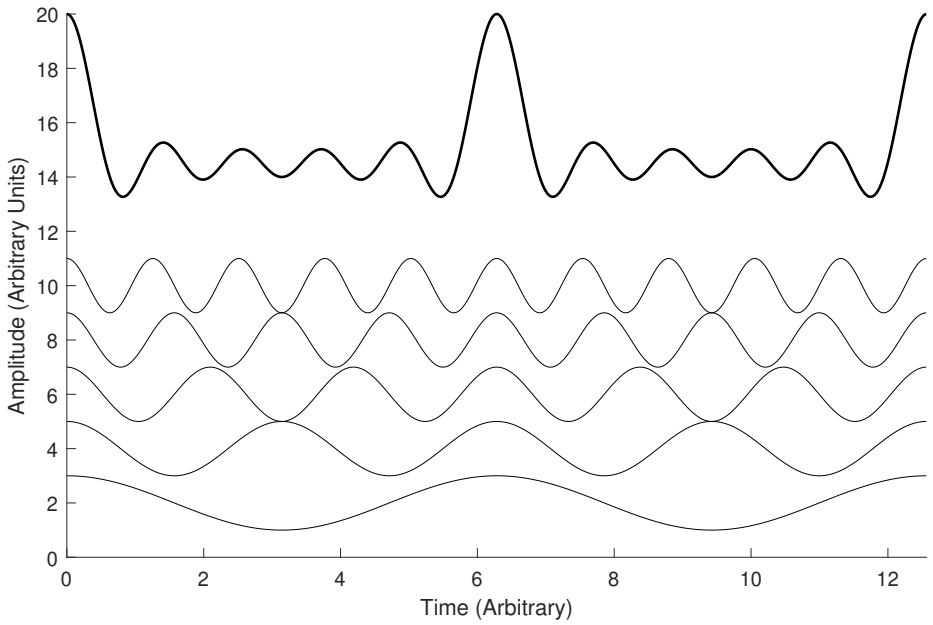


Figure 1: Due to the superposition principle, five individual waves sum to create a combined wave. Note: This figure is provided purely to show the *shape* adjustment due to constructive and destructive interference; the reader will observe that the summed wave has been translated downwards in order to make the point clearer.

A short pulse is created by taking a broad range of frequencies and manipulating them in such a way. With these basics explained, the focus moves onto the central theme of this thesis – High-order Harmonic Generation (HHG).

1.1 High-order Harmonic Generation

HHG is the generation of harmonics (multiples of driving laser frequency), with an emphasis on the higher harmonics, whose upper energetic limit is called the 'cut-off'. HHG is only possible because of a particular setup, wherein infrared (IR) light (supplying the electric field) is

focused to reach a high intensity, such that it rivals the atomic potential.

A standard setup when performing an experiment using HHG would include the apparatus shown in figure 2.

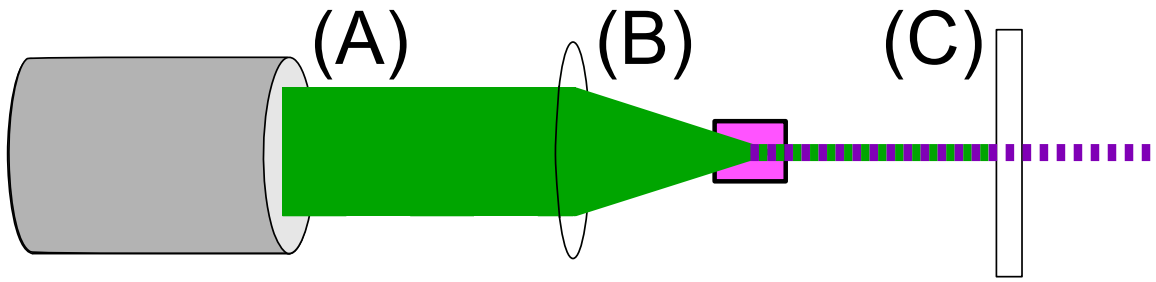


Figure 2: The standard components required to create HHG. (A) is a laser source, with produced light shown as a green line for visual clarity (B) represents the focusing into the target medium and (C) represents the 'control and filter' stages.

Stage (A) is usually an IR laser within the range of 800 nm - 1100 nm, emitting an ultra short pulse of between 20 - 200 fs duration. The laser in the above image for example could be a Ti:sapphire laser, which supports a large bandwidth, however other optical components in the laser path can further limit this bandwidth.

Stage (B) is an initial control of the IR light, such that it is focused into what is traditionally a container of gas (the pink box in the figure), a gas jet, or can even be done with a solid target [1]. The IR light needs to be focused like this in order to obtain the intensities required to match that of the atomic potential. Outside of the medium container will be vacuum, which ensures that the high frequency Extreme Ultra Violet (XUV) photons are not absorbed on their way to the destination.

Stage (C) is used for filtering out the IR pulse, so that the useful XUV light remains. This XUV light can then be manipulated and controlled by further mirrors and gratings. It is not trivial to control the light with mirrors – one can only expect to do 'simple' manipulations.

The standard setup will not be in a straight line as shown above, and the figure does not resemble the real life scale – there are also usually several reflections and refractions between these stages in any real life setup to manipulate the beam required for the experiment.

Stage (B) in figure 2 houses the atomic medium, which is where the HHG process takes place. The IR light will interact with the atoms in the medium, and the model used in this thesis takes the semi-classical approach to explain it.

1.1.1 Semi-Classical Model

HHG can be explained semi-classically using the Three Step Model, which breaks down the process into three steps:

1. Ionisation, through tunnelling of the electron
2. Propagation, where the electron propagates freely in the electric field
3. Recombination, resulting in the transmission of a photon (with energy dependent upon the kinetic energy of the returning electron)

The process can be explained simply by figure 3.

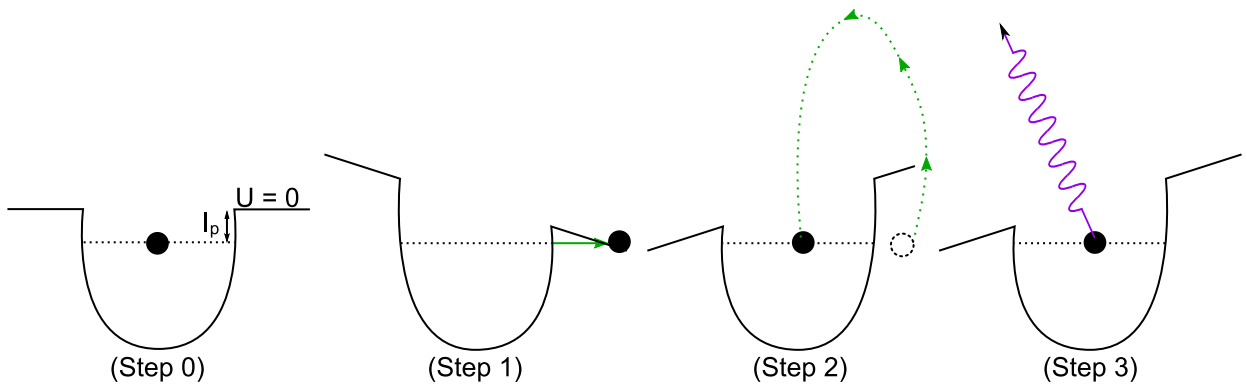


Figure 3: The three step model explained. (Step 0) shows an atom with no bias applied. The other steps line up with the three step model. Note that the ionisation potential, I_p is a value unique to the atom from which the electron will be liberated.

This process has been portrayed in many different ways, throughout many different theses and papers and has reached the point where it is considered "textbook" for the average attosecond physicist [2].

Using figure 3 to explain the three different stages in more depth:

The setup will consist of a laser, usually IR, which will be focused into a medium containing an element – typically a noble gas, although solid targets are also used. Step 0 gives an oversimplified view of how the average atom looks at ground state energy (without an external field). In Step 1, the IR laser is focused on the sample, which provides an intensity that must be in the same ballpark as the atomic potential to be effective – this is why it is so important that the pulses are ultra short, as intensity is inversely proportional to time. The ionisation potential (I_p) that can be seen in small font in Step 0, is an energy that is unique to each atom – it is the energy required to extract the least stable electron from the atom. The electric field lowers this ionisation energy sufficiently that the electrons have a significant probability to tunnel.

Once an electron tunnels, the process can be treated classically (although in reality it is more accurate to describe it as an electron wave packet (EWP)). The electron will now follow a specific trajectory while outside of the atom, which is dictated by the external laser field. The trajectory of the electron can be calculated by taking equation (1) and substituting this into Newton's second law, $F = ma$.

Noting that $F = -eV$, where $-e$ is the charge of the electron and V is potential energy, a substitution is made, providing equation (2), which relates the acceleration of the newly freed electron to the laser field.

$$a \quad = \quad \ddot{x} \quad = \quad -\frac{eE_0}{m_e}\sin(\omega t) \tag{2}$$

With the acceleration defined, we can then take a step further and integrate, which provides equation 3, the velocity of the electron at a time t after ionisation at t_i. An assumption is made that the electron has no velocity upon initially tunnelling.

$$v \quad = \quad \dot{x} \quad = \quad \frac{eE_0}{m_e\omega}\Big(\cos(\omega t) \; - \; \cos(\omega t_i)\Big) \tag{3}$$

Taking the final logical step, another integration is done (noting that t_i is a constant, meaning that the second term in equation (3) is a constant in the process of integration). This leads to equation (4) and the ability to tell exactly where the electron is at all times. This is based upon the assumption that the tunnelling is sufficiently close to the atom that the initial displacement is set to 0, and the initial velocity will be 0.

$$x \quad = \quad \frac{eE_0}{m_e\omega^2}\Big(\sin(\omega t) \; - \; \sin(\omega t_i) \; - \; \omega(t-t_i)\cos(\omega t_i)\Big) \tag{4}$$

This equation is arguably the most important one for the investigations done in this thesis, and is the basis for all of the results in section 3.

The reason that this equation in-particular is so important is that it, along with its derivatives, allow for a complete knowledge regarding the electron and all its attributes. At any single time, t, the electron's current position, velocity, and acceleration can all be calculated – and by extension, the calculation of the velocity of the electron (equation (3)) allows us to easily find the kinetic energy of the electron by a substitution, as performed in equation (5).

With this in mind, from a simulation point of view, the trajectories can be coloured based upon the "recombination energy" that they encompass.

$$E_{\text{kin}} \quad = \quad \frac{1}{2}mv^2 \quad = \quad \frac{1}{2}\frac{e^2E_0^2}{m_e\omega^2}\big(\cos^2(\omega t) \; - \; \cos^2(\omega t_i)\big) \tag{5}$$

The cycle-average of the electron's kinetic energy is taken, which is given its own name: *ponderomotive energy*, symbol U_p. In the literature, this is often expanded upon by noting that $\omega = 2\pi c/\lambda$ [3, 4].

The figure following will be shown in terms of the electric field strength, and the ponderomotive energy will then be defined as equation (6). As mentioned, ponderomotive energy is the *cycle-averaged* energy of an oscillating electric field, which means taking the expected value of the squared cosine. This results in an extra factor of 2 in the denominator, leading to

$$U_p \quad = \quad \langle E_{\text{kin}} \rangle \quad = \quad \frac{1}{2}\frac{1}{2}\frac{e^2E_0^2}{m_e\omega^2} \quad = \quad \frac{e^2E_0^2}{4m_e\omega^2} \tag{6}$$

When considering the energy gained along the different trajectories by the electron, the classical approach results in a maximal potential gain of $3.17U_p$ – which can be verified by plotting all of the different trajectories and their respective energies, and then finding the peak amplitude of the electron kinetic energy. This is shown in figure 4 on the y-axis.

The energy calculations done for this figure have been based only on classical trajectories, however quantum effects can be included by making adjustments to the equation – which have been performed by Lewenstein *et al.* [5].

One can predict the highest harmonic order that the HHG process can produce semi-classically, with the cut-off law, given by equation (7). When performing calculations of this nature, it is important to consider that the electron is not going to be in-vacuum once released – therefore this is a theoretical cut-off not usually reached in experiment.

$$E_{\text{photon}} \quad = \quad qhf \quad = \quad 3.17U_{\text{p}} + I_{\text{p}} \tag{7}$$

where q is the harmonic order (integer), h is the Planck Constant, f is the driving frequency of the laser, U_{p} is the ponderomotive energy and I_{p} is the ionisation potential of the gaseous/-solid target.

The left hand side represents the energy of the produced XUV photon, and of the two terms on the right hand side, $3.17U_{\text{p}}$ is the maximum attainable kinetic energy of the electron while outside of its parent atom, and I_{p} is the ionisation energy that will be gained due to the atom re-accepting the electron. This a bit less simple when tunnelling is involved, but if thought of classically, it costs energy to remove an electron from an atom and provides energy to return it, provided that the atom is made more stable by accepting it.

Considering an arbitrary atom with one electron, and allowing the trajectory to be purely classical, we arrive at figure 4. A one electron system is a good model for a many-electron system, as the ionisation potentials of different electrons in a many-electron system will all have varying potentials. As an extension of this, noble gases (such as Neon or Argon) are among the most frequently used target mediums and have a full outer shell.

There are two factors involved in withdrawing only one electron – the first is that the intensity of the ionising light is on the same magnitude as the atomic potential of the atom, so tunnelling becomes likely. The second is when the IR laser interacts with the atom, one electron will get slightly perturbed before the others, which increases its likelihood of being the one that will be ejected. The other electrons that haven't been perturbed (as much) will be bound more tightly than the perturbed one.

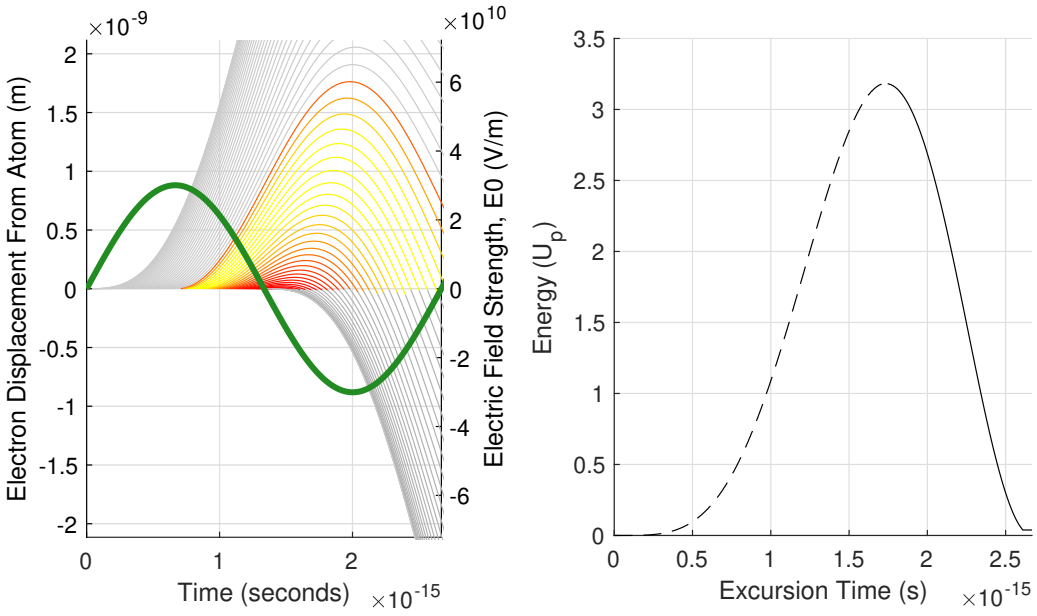


Figure 4: Left: A number of possible trajectories for an electron ejected within the first half-cycle. Right: The energies of returning electrons, as a function of their time outside of the atom. Grey trajectories indicate electrons that do not return to the parent atom. The dark green line shows the strength of the IR electric field as time progresses (with units along the right hand y-axis of the graph). In terms of trajectory line colour, yellow represents the most energetic trajectories, while as red represents the lower-energy trajectories. The grey trajectories in the *positive*-y axis are trajectories of electrons that never return from the initial ionisation, while as the *negative*-y axis grey trajectories are electrons that complete at least one cycle before being expelled by the electric field.

In figure 4, the right image represents a continuum of possible kinetic energies that the electrons can have upon returning to the parent atom – the image represents one cycle of the IR laser. As was mentioned in the preceding introduction of ponderomotive energy, it can be seen that the maximum electron kinetic energy is 3.17 U_p. The quadratic shape of the graph indicates that there is more than one path sharing the same kinetic energy. These different paths are the different possible trajectories that can be taken.

The dotted line shows the "first" time that each energy is reached, and has been designated the logical name – the "short" trajectory (as the electrons have a lower excursion time than any other trajectory). Further to this, the solid line following it is known as the "long" trajectory. Multiple passes of the electron (should it not recombine upon returning) are examined further in the thesis, however the above image only shows the short and long trajectories. After the short and long, trajectories are labelled by number (ie: third, fourth, fifth, etc).

1.1.2 SFA/Lewenstein Model

Although the approach of this thesis is based upon the three step model, there are more angles from which the phenomenon of HHG is approached. Modern approaches tend to view HHG through the lens of the SFA (Strong Field Approximation, used in Lewenstein's model [6]), and through the TDSE (Time-Dependent Schrödinger Equation). It is noted by Lewenstein that use of the TDSE tends to be computationally expensive [5].

In their paper, Lewenstein and colleagues note that a particular problem with the three step model arises from the cut-off energy (equation (7)), which although a good approximation,

appears to not be completely accurate experimentally (the difference is explained in Lewenstein's paper as being due to "propogation effects" [5]).

The paper improves upon the three step model by introducing quantum mechanics, which allow for the discrepancies between theory and experiment to be smaller. The theory is sometimes referred to as the 'SFA Model' because of the assumptions made in the development of the theory. These assumptions [5, 6, 7] are

1. The electron, once outside of the atom, has a trajectory that is not influenced at all by the atomic potential.
2. The atom can only be described by one bound state. This means in essence that all/any contribution from the excited bound states can be ignored. In other words, the atoms that make up the medium are assumed to all be in the same state, which has a set ionisation potential much greater than that of the laser.
3. Atoms cannot be completely depleted of electrons – most electrons typically remain, as electrons are only stripped typically at the peak of the electric field strength.

Lewenstein *et al*. in his paper goes on to break down these restrictions and explain where they can be relaxed, why they are needed, among other notes [5].

The central differences between the more advanced theory and the one used in this thesis, is that the semi-classical approach appears to be called 'semi-classical' for the fact that it qualitatively requires quantum mechanics in order to explain the ionisation and recombination, however does not have within its repertoire the ability to explain interference, or how a wave packet propagates due to quantum effects.

An example of why it is important to take into account these effects, is because several different quantum paths can produce the same EWP energy, and these different paths can interfere with each other [8].

In contrast with the semi-classical model, the SFA model treats the electron as an EWP, which at the time of recombination produces not one single energy, but the whole harmonic spectrum of solutions at once. The SFA is good at finding trajectories, however requires the assumptions previously mentioned. The SFA ignores the atom (and by extension: atomic potential) completely and treats the electron as if it is travelling freely within the electric field.

It can be summarised by saying that the classical model allows for greater intuition, but misses some of the subtleties [3].

1.2 Phase

1.2.1 Introduction To Phase

When working with more than one wave, it is important to be aware of how different frequency waves behave in combination with each other. The HHG process takes place in many atoms at the same time and thus the emitted XUV photons are numerous in quantity. Each of these photons have their own phase value, which is an important quantity to be able to calculate. The following section will delve into the phase of these photons and how to calculate it.

There is an SFA approach to phase, which extends beyond the simple semi-classical picture through use of Feynman path integrals and quantum orbital numbers. It leads to equation (8). This equation, although not used in this thesis, calculates phase through the integration of EWP momentum.

$$\Phi_j^q([r_j(t_i,t_r,\mathbf{b})]) \quad = \quad q\omega t_r \; - \int_{t_i}^{t_r} \Big(\frac{(\mathbf{p}+\mathbf{A}(t))^2}{2} + I_p \Big) \mathrm{d}t \tag{8}$$

The breakdown and use of this equation can be observed in the paper by K. Varjú *et al*. [9], however this thesis will use an approximation of the integral, suggested by Chen Guo [10]. This approximation is based upon dipole-phase and obtains a good match with the integral above.

In order to introduce and justify the approach suggested by Chen, it is important that the distinction between phase delay and group delay be made first.

1.2.2 Choosing The Type of Delay

A delay is simply the amount of time between two points, however one of the important distinctions that needs to be made when talking about delay in optics, is how this delay is to be quantified.

In a world where all frequencies travel at the same speed through all mediums, this wouldn't need to be discussed, however the two types of delay to choose from are 'phase' delay, and 'group' delay. In order to facilitate an explanation of the difference, equation (9) is provided.

$$\Phi \quad = \quad \xi(\omega)\,\omega \tag{9}$$

Here, $\xi(\omega)$ is a function relating phase to angular frequency. What this means is that the value of $\xi(\omega)$ decides whether different frequencies are treated differently in terms of their phase.

Before continuing this explanation, it is important to also know the mathematical definitions of phase delay and group delay, given by equation (10) and (11) respectively.

Phase delay is simply the number of cycles, for a given frequency, between two different points in time. Group delay has its focus on the envelope as a whole rather than individual points, while as phase delay would indicate that all frequencies are delayed equally.

$$\text{Phase Delay (P.D)} \quad = \quad \frac{\Phi}{\omega} \tag{10}$$

$$\text{Group Delay (G.D)} \quad = \quad \frac{\mathrm{d}\Phi}{\mathrm{d}\omega} \tag{11}$$

A few different scenarios can take place here, depending on the form that $\xi(\omega)$ takes.

Scenario 1 - $\xi(\omega)$ is a constant. This means that the relationship between phase and frequency is linear. As equation (12) demonstrates (by setting $\xi(\omega) = 2$ as an example), the differentiation would result in only a constant remaining, meaning that all points on the envelope would be adjusted equally. As can be observed by using equation (10), this means that the group delay is the same as the phase delay, meaning that the resultant shift is a translation of the wave, with the sum wave retaining its shape.

$$\text{G.D} \quad = \quad \frac{\mathrm{d}}{\mathrm{d}\omega}\big(2\omega\big) \quad = \quad 2 \tag{12}$$

Scenario 2 - $\xi(\omega)$ is not a constant, but rather a variable defined by a function. This scenario is more accurate within the field of HHG. The experimentally observed relationship between phase and frequency in HHG is non-linear [9] and hence it is important to take this into account. To demonstrate this in the same manner as before, an arbitrary non-linear $\xi(\omega)$ is chosen to make this demonstration ($\exp(2(2\omega+3))$), as demonstrated in equation (13). Calculating the phase delay will provide a result that is not equal to the group delay.

$$\text{G.D} \quad = \quad \frac{\mathrm{d}}{\mathrm{d}\omega}\big(\omega\exp(2(2\omega+3))\big) \quad = \quad (4\omega+1)\exp(2(2\omega+3)) \tag{13}$$

These two scenarios have been coded into MATLAB, in figure 5. Scenario 1 is the central panel, and scenario 2 is the right panel.

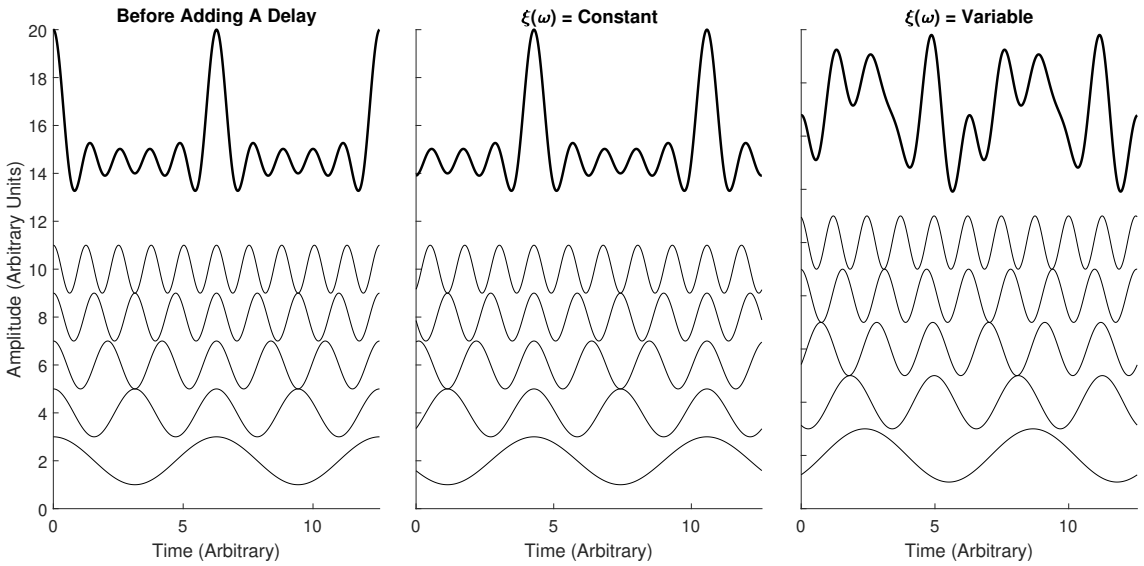


Figure 5: Left: Superposition involving 5 waves with no delay applied. Centre: Same setup, but all frequencies delayed by an equal amount. Right: Same setup, but with all frequencies delayed by varying amounts. The MATLAB code to reproduce this is provided in the appendices.

It can be seen in the figure that, depending on the form of $\xi(\omega)$, the envelope either retains its shape or it deforms. This is the reason that it is important to know the relationship between phase and frequency. In other words – it is important to have some sense of what $\xi(\omega)$ is.

In thinking about how phase might be adjusted with time, there is an intuitive sense that the recombination time, t_r relates to the group delay through the trajectories, as the frequencies can be seen changing continuously over small regions of time - as demonstrated in figure 6.

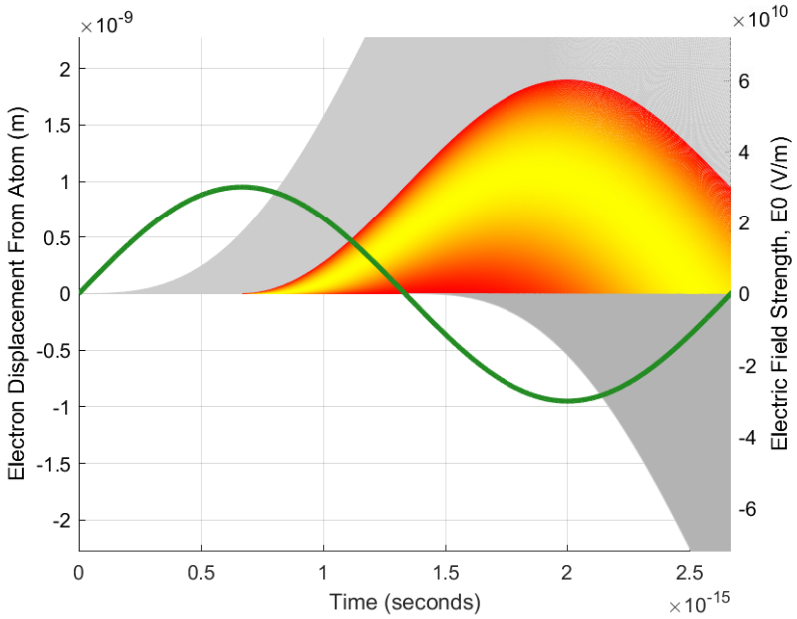


Figure 6: It can be seen in only half of one IR cycle (dark green), that the XUV photon frequency experiences the full range of lowest frequency (red) to highest frequency (yellow).

Although presented purely based upon intuition, this does allow for a good approximation of the SFA approach, as presented in the next section.

1.2.3 Semi-Classical Approximation of Equation (8)

Chen provides a way to get the approximate results, which can be done as follows:

1. Ascertain a relationship between t_r (where t_r is used to approximate group delay) and frequency (made by approximating the energies with a linear fit)
2. Integrate the area under the each trajectory
3. 'Stitch' them together in such a way that the phases at the intermediary points between trajectories is continuous.
4. Multiply the result by 2π, noting that group delay is defined based upon angular frequency.

The result of using this approximation is figure 7, which has a good tie with figure 2a of 'Frequency chirp of harmonic and attosecond pulses', a paper by K. Varjú *et al*. [9]. The constants/settings were made to match the reference study, with the following setup:

```
Laser Wavelength: 800 nm
Laser Intensity: 1.2e18 W/m^2
Atom: Argon
```

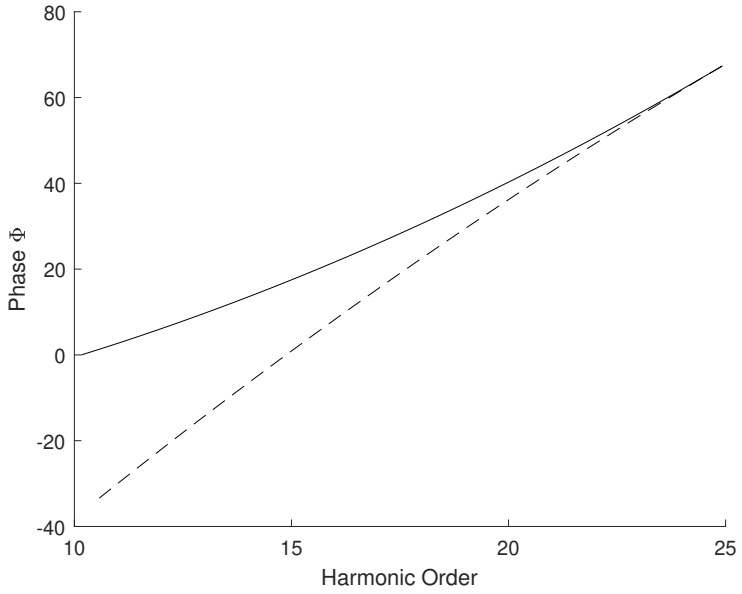


Figure 7: The result of following the procedure shown at the beginning of this section. The solid line is the short trajectory, and the dashed line is the long trajectory.

As was mentioned briefly in section 1.1.2, there is a difference between the maximum cut-off energies, wherein the semi-classical version displays circa 25 as the cut-off, while as the reference displays 30.

One can also relate laser intensity and phase, as seen in figure 8. This matches up reasonably well with figure 1a of the reference study.

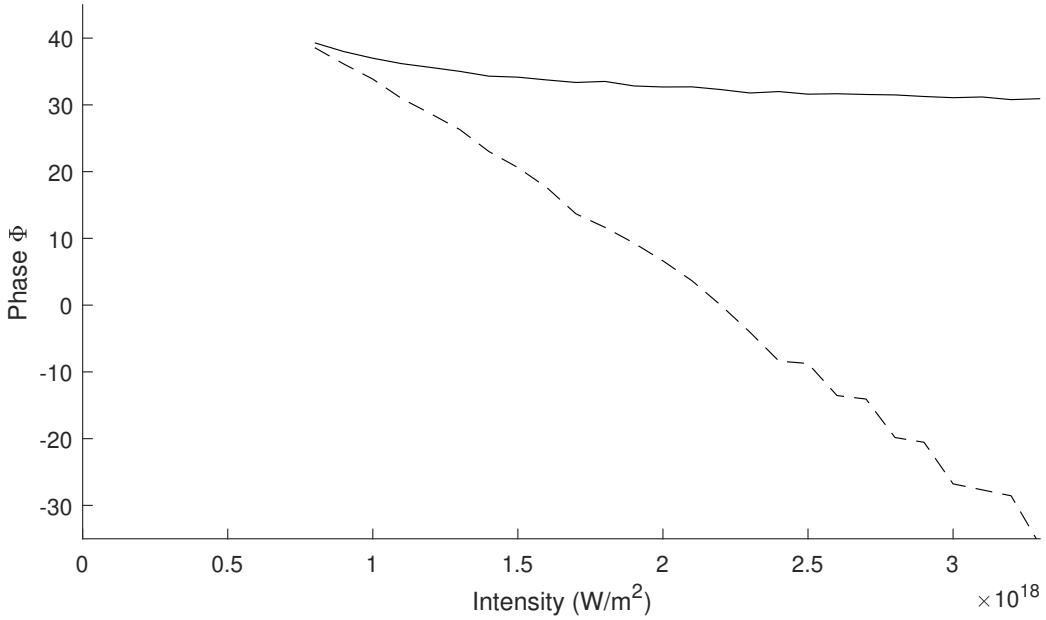


Figure 8: The 19th harmonic of Argon at different laser intensities. The method allows for a good approximation to the reference study. The solid line is the short trajectory, and the dotted line is the long trajectory.

Experimentally, one can take additional measures to reduce the discrepancy in results from

the theory by phase matching [3]. In essence, phase matching consists of taking into account several additional factors such as the Gaussian profile (Gouy phase shift), shift due to mismatch of the dipoles, mismatch due to dispersion of the gas and mismatch due to plasma dispersion. This thesis only mentions these and does not go into depth on this topic.

1.3 Fourier Transformations

An important process that will be utilised throughout the thesis is that of the Fourier transformation. Initially, the time-frequency Fourier transform will be explained, and then the focus will move to the *spatial* version of the transform, which is the one that will be used.

The HHG process all happens within the focus of the IR light, however the XUV light output will be measured up to several metres away on a detector. To retrieve how the shape will look on the detector, the emitted photons are Fourier transformed. The thesis up to this point has been investigating the 'immediate' photon output (known as 'near-field'), however will now change scenery slightly to look at how the output light will behave once it has propagated (known as 'far-field').

The Fourier transform does this by switching between two different domains in order to get information about a repeating function. Imagine a situation where there is an audio clip playing of a person talking, but there is an irritating high-pitched noise in the background. A Fourier transform allows you to sample the clip and find out where this annoying noise is coming from (by informing you of its frequency and amplitude). You might then reconstruct the audio clip with this frequency filtered out.

There is a continuous version defined as an integral, and a discrete version defined as a discrete sum. The discrete version will be explained here due to its pedagogical nature, however the continuous version works on the same principle.

1.3.1 The Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is the version of the transform where points are sampled at discrete points, making it simple to program (although time-consuming in terms of processing time). As long as there is a manner in which the wave can be regularly sampled, then a DFT can be taken.

The DFT is given by equation (14).

$$X_k = \sum_{n=0}^{N-1} x_n \cdot \exp\Big(-i2\pi kn/N\Big) \tag{14}$$

where X_k represents the kth frequency sample, N is the number of samples taken, x_n is the value of the amplitude at the nth sample. k is a counter for the current frequency being used, and n is a counter for the current sample being used.

Pedagogically, one can think of this formula as checking how much 'agreement' there is between all of the different terms in each X_k sum. For example, if a frequency is found which is absolutely required to produce the wave, then the different terms of the X_k sum will add constructively. If the frequency is not present at all, then all of the terms in the particular X_k sample will destructively interfere to produce a value of 0.

An appendix has been created, section 6.1, which goes into further detail about the process that was used to program the DFT. In this appendix, step 4 is where the Fourier transform takes place. To demonstrate this process, two waves have been combined in figure 9 below. The wave in this figure is the result of expression (15).

$$\sum_{\alpha=1}^{2} \alpha \cos(\alpha \omega t) \tag{15}$$

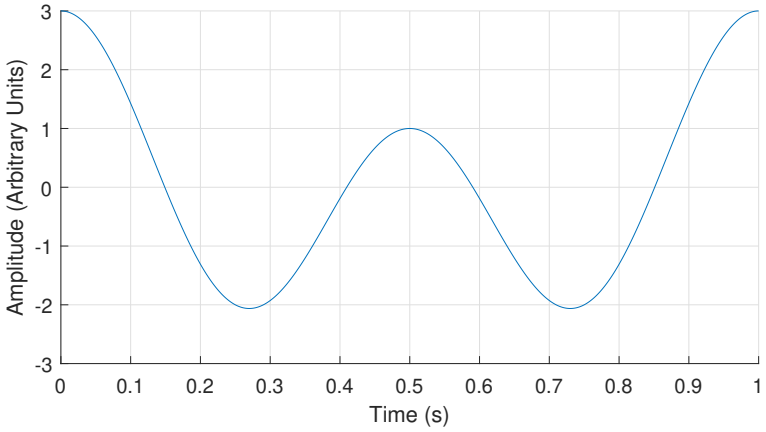


Figure 9: A visual representation of the wave in expression (15)

After samples at equidistant points have been taken, these are noted as the x_n values. A value of k is then taken, and cancellations/constructions will take place depending on the "tuning variable", which is the exponential factor.

To give an example using figure 9, $k = 0$ will look as per equation (16), for N=5.

$$X_0 = x_0 e^{\frac{-i2\pi \cdot 0 \cdot 0}{5}} + x_1 e^{\frac{-i2\pi \cdot 0 \cdot 1}{5}} + x_2 e^{\frac{-i2\pi \cdot 0 \cdot 2}{5}} + x_3 e^{\frac{-i2\pi \cdot 0 \cdot 3}{5}} + x_4 e^{\frac{-i2\pi \cdot 0 \cdot 4}{5}} \tag{16}$$

Each of the n terms has an opportunity to increase or decrease the value on the right hand side of the equation, with the x_n value contributing to the sign and amplitude.

For the next k in the sequence, equation (17).

$$X_1 = x_0 e^{\frac{-i2\pi \cdot 1 \cdot 0}{5}} + x_1 e^{\frac{-i2\pi \cdot 1 \cdot 1}{5}} + x_2 e^{\frac{-i2\pi \cdot 1 \cdot 2}{5}} + x_3 e^{\frac{-i2\pi \cdot 1 \cdot 3}{5}} + x_4 e^{\frac{-i2\pi \cdot 1 \cdot 4}{5}} \tag{17}$$

A single value of X_k takes information about all of the amplitudes spread throughout the sample, and because of the presence of the imaginary unit (i), can even "cancel out" the negative sign of the amplitude. In the event that this happens, a frequency plot can be created, which provides information about the wave. The Fourier transform of figure 9 is given in figure 10.

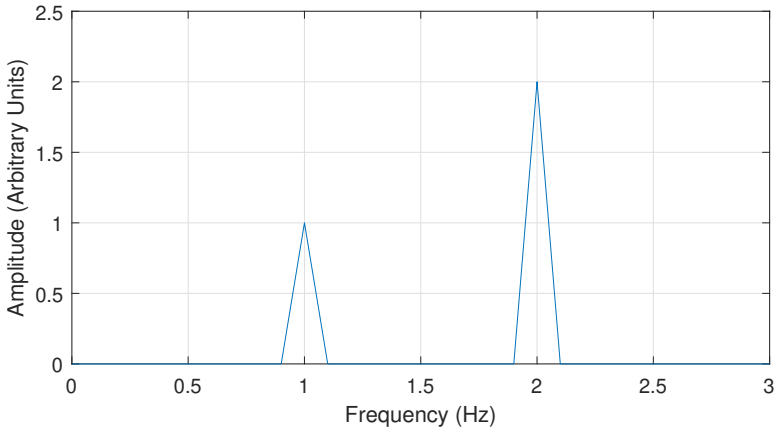


Figure 10: The Fourier transform of figure 15, with a time span of 10 seconds, and N=500.

One can improve the fidelity of the Fourier transform by increasing the time that the sample is taken, however the number of samples must be sufficient that the Nyquist Limit is not crossed, or the results will be partially meaningless.

1.3.2 The Shannon-Nyquist Sampling Theorem & Aliasing

The Shannon-Nyquist Sampling Theorem was the child of two titans within the field of communication and telegraphing theory (Harry Nyquist and Claude Shannon), whose papers [11, 12] gave the subject a more rigorous mathematical basis.

The result was the expression of a limitation on the frequencies found through the Fourier process, which is known as the *Nyquist Limit*.

In short, if there are frequencies of interest which are greater than half of the sampling frequency, there will be a distortion within the signal known as *aliasing* and if the Nyquist Limit is not respected, then these frequencies will not be visible at all. In the field of signal processing, this means that the bandwidth (highest frequency) needs to be chosen specifically to prevent distortion, by ensuring that the Nyquist Limit is greater [13].

For the sake of demonstrating this, figure 11 is the plot of expression (18).

$$\sum_{\alpha=1}^{3} \alpha \cos(\alpha \omega t) \tag{18}$$

While the expression itself is plotted in black, the sampling is shown in red, which appears to record a completely different wave profile.

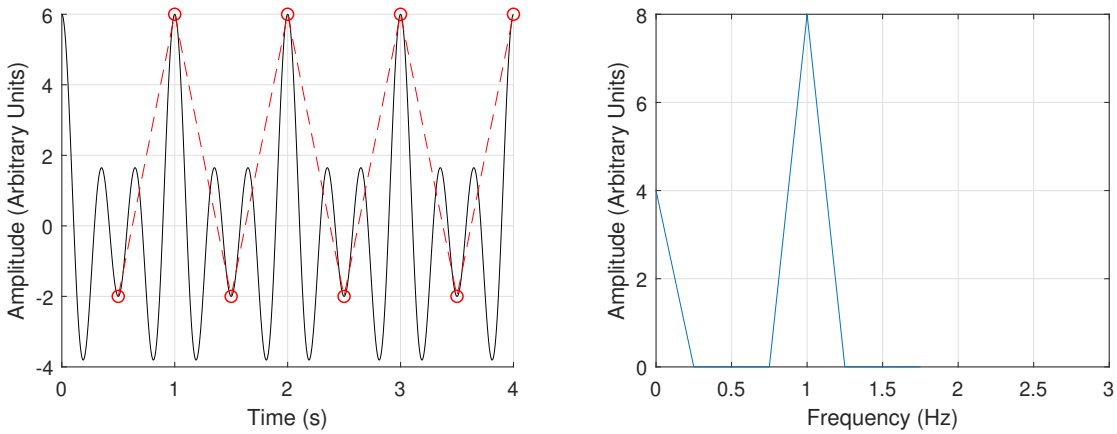


Figure 11: The Fourier transform of the wave of expression (11). 8 samples have been taken, while as 24 samples are the indicated minimum required to pass the Nyquist Limit. The red indicates the sampling.

An extreme-case example of this can be seen by taking a simple wave: $2\cos(2\omega t)$. If a 4 second time-period of this wave is plotted, and one attempts to Fourier transform it with too few samples (half the required number, for example), then figure 12 arises.

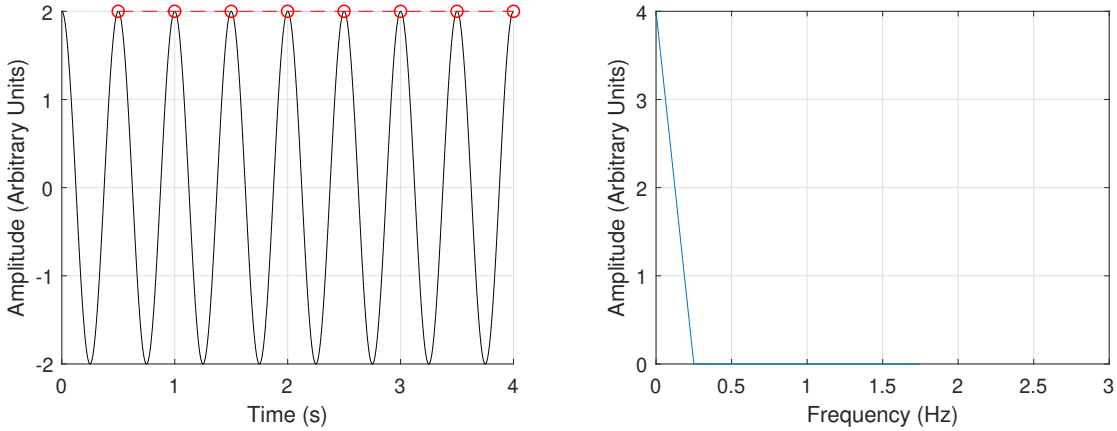


Figure 12: The Fourier transform of a $2\cos(2\omega t)$ wave. 8 samples have been taken, while as 16 samples are the indicated minimum required to pass the Nyquist Limit. The red indicates the sampling.

This is an extreme case, but it demonstrates why observing this limit is important.

Even if one adheres to the limit precisely, the resultant plot may display the correct frequencies, but appear to have a symmetry. A good way to demonstrate this is by re-sampling figure 11, but with the sampling frequency at the Nyquist Limit. This is done in figure 13.

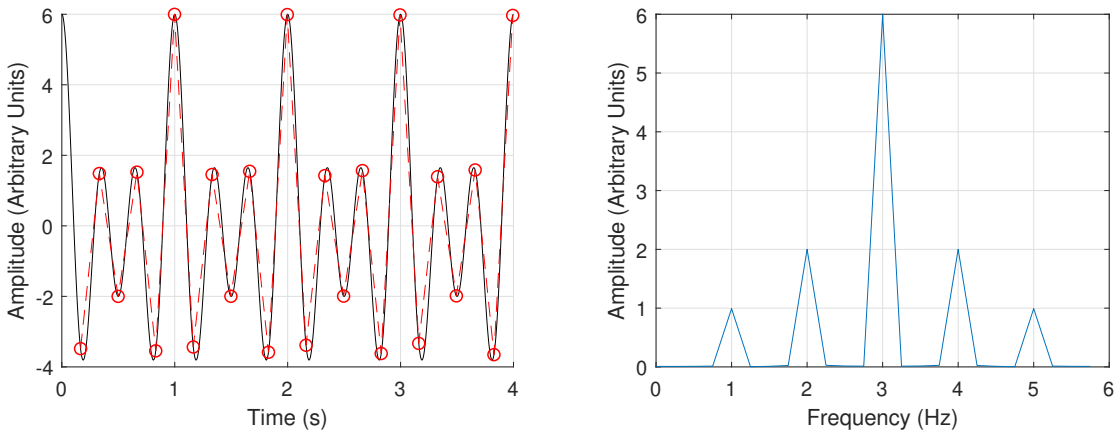


Figure 13: The Fourier transform of expression (18). The sampling frequency has been set to the Nyquist Limit. Symmetry can be observed.

The amplitude for the 3 Hz peak is incorrect – It is double what it should be. This is an artefact of Euler's formula, as the plotted wave was the sum of cosine waves, and $\cos(x) = \frac{1}{2}\left(e^{ix} + e^{-ix}\right)$. The frequencies past the Nyquist Limit are "negative frequencies", and represented by the second term in Euler's formula. These can therefore be ignored – however if one wishes to get the correct amplitude for the frequencies present, one should proceed a good distance beyond the sampling frequency.

1.3.3 The Fast Fourier Transform (FFT)

In reality, the DFT is costly in terms of computation for a large number of samples. For this reason, an algorithm often used is the FFT (Fast Fourier Transform), which in several programming languages already has a defined module for the user (*fft* in MATLAB, *scipy.fft* in Python, etc).

The previous subsections have gone into reasonable depth describing both the process and limitations of the Fourier process, so the specifics of the FFT are not covered here.

What the FFT does is take advantage of the fact that points over the Nyquist Frequency can be disregarded, and performs an elegant algorithm including complex numbers. The exact process of the algorithm is covered by numerous videos, but here it is enough to know that the MATLAB function FFT() is used, instead of the author's own DFT function.

1.3.4 The Spatial Fourier Transform

The spatial Fourier transform is the distance analogue of the traditional Fourier transform, where distance is the analogue of the time domain, while as what is referred to as "spatial frequency" is the analogue of the frequency domain. The idea behind doing a Fourier transform of distance is, in effect, that one can see the underlying characteristics of the image.

These characteristics mentioned manifest when the source is sufficiently far from the canvas that the waves can be treated as 'planar' in nature. When the light source reaches a target as one planar wave, it can be classed as being in the 'far-field'. A good way to demonstrate this is to first show two samples, which have a different number of slits present. In order to make it clear of the setup in question, figure 14 has been provided.

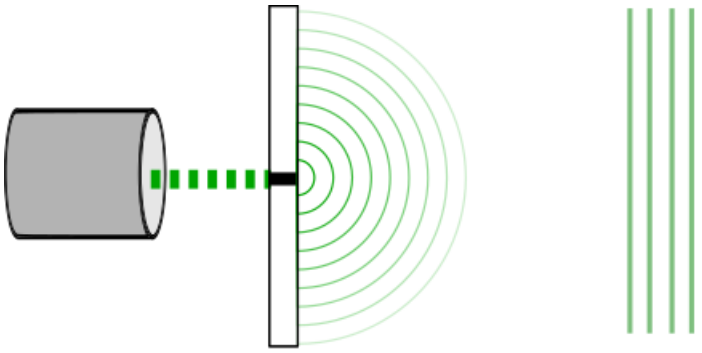


Figure 14: On the left, the near-field source is seen, with waves spreading upon reaching the aperture. On the right, far-field waves are seen propagating in a plane.

It is worth mentioning that the experimental situation is more complicated than figure 14, as the figure would indicate no interference. The reality is that there will be interference, due to Huygens' Principle.

A way to gain intuition as to why the Fourier transform relates the near-field to far-field can be seen in figure 15. The green light approaching the aperture has a continuous wave-front, wherein all points are a source of wavelets.

If one considers a simple scenario, the breaking down of the aperture into four sources of wavelets, then interference results. This can be seen in the image if one looks horizontally to the centre of the aperture, where all four semicircles can be seen to be lining up.

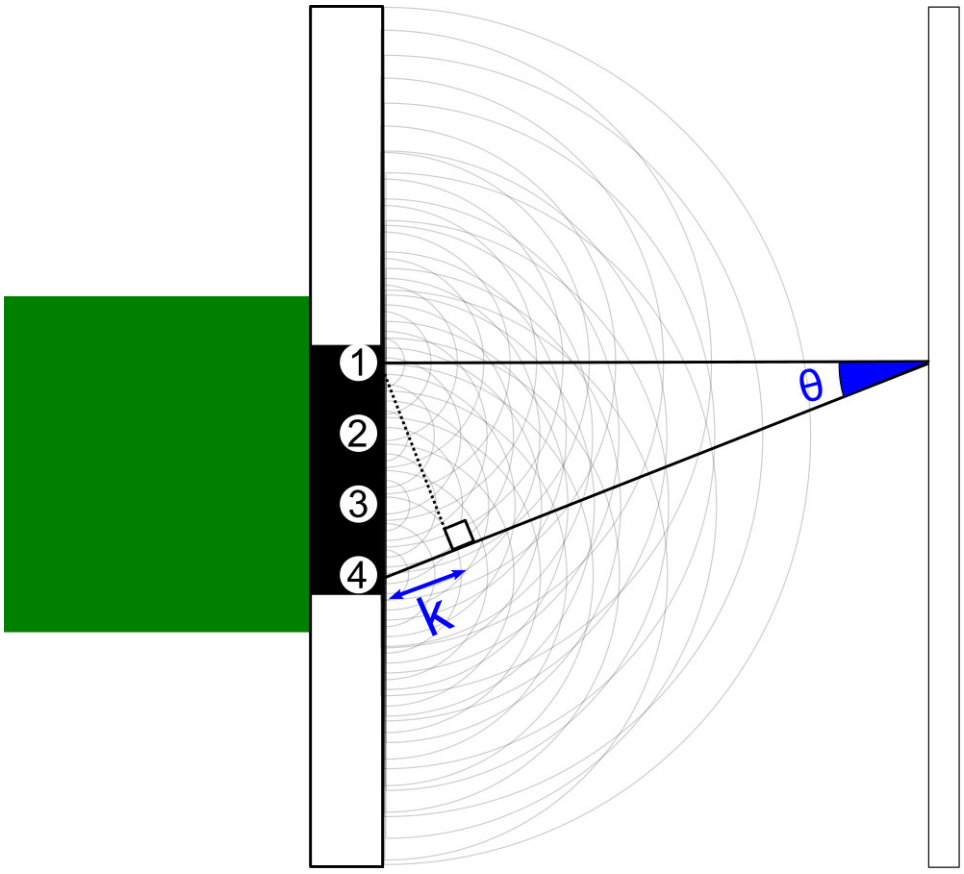


Figure 15: An expansion of figure 14, wherein four Huygens' wavelets are considered. Path difference is added to facilitate discussion.

As seen in the figure, there is a path difference, k, between the wavelets resulting from '1', and from point '4'. This path difference contributes to the difference in phase, based upon how this path difference compares to the wavelength. Following through section 1.3.1, it is this path difference that contributes to the phase in the exponent. The resulting interference pattern forms now as a sum of all the different path contributions, over all angles of θ over the far-field.

Experimentally, the result of using a one-slit setup is shown in figure 16.

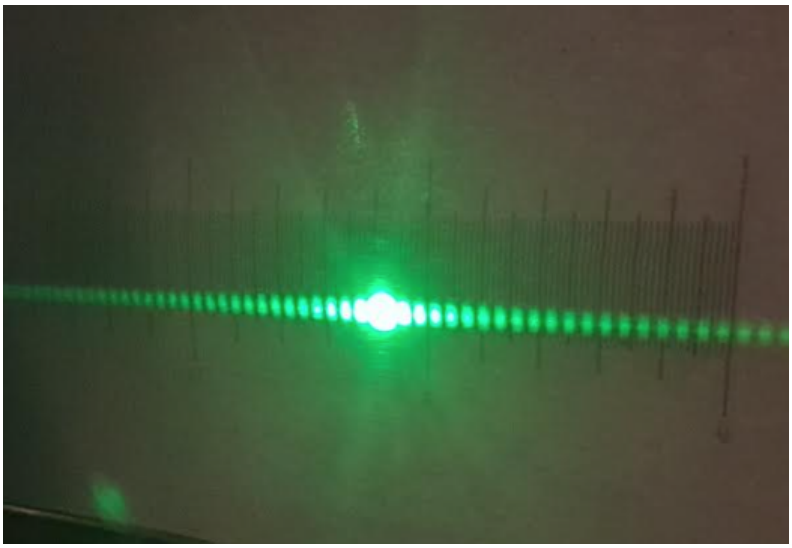


Figure 16: Far-field imagery with a single slit source (the experimental version of figure 14).

In order to demonstrate the programming analogue of the left image of figure 16, a similar setup has been programmed into MATLAB, producing figure 17. The similarities are clearly visible and show the clear benefit of the spatial Fourier transform in replicating the far-field output.

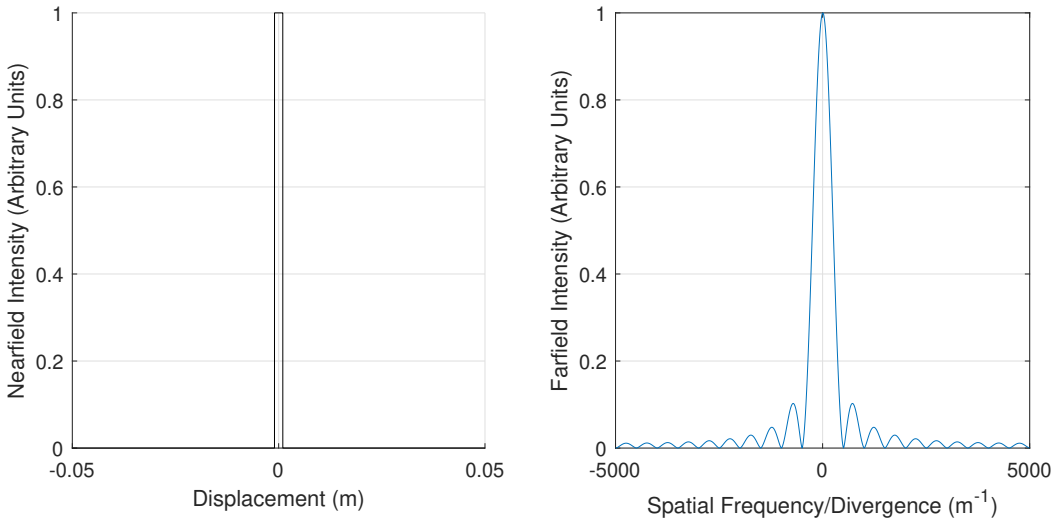


Figure 17: Demonstration of a Fourier transform of a 10cm long board, with a 2mm wide slit in the centre. Left demonstrates the slit, and right is the Fourier transformation.

It is then clear that one can gain knowledge about the far-field characteristics of an XUV photon by analysing its near-field characteristics.

2 Method

2.1 Simulation Code

The simulation code of this thesis is written in MATLAB. The basic code began with simulating all possible trajectories of a single electron, and grew by adding layers of complexity to the simulation. On top of the basic code, built-in commands such as "FFT" (Fast Fourier

Transform) were used.

Initially, the code was written with a "plot and overwrite/forget" method, however it quickly became apparent that this was a bad decision for numerous reasons. As a consequence, the code was re-written and a function-based code was used instead.

The raw code can be found in section 6.3, including comments to explain different sections and what the intention was. Section 6.2 offers an explanation and breakdown of all of the different functions used in the code.

The functions work in combination with one-another, so for example if the requirement is to produce a table of all of the possible energies stemming from a laser of intensity I, then the code would be written as follows;

```
Matrix = DisplacementMatrixMaker(Time, EjectionTimes, q, E0, m_e, w_0);
Table = BasicEnergyTableMaker(Matrix, lastEjectionTime, q, E0, m_e, w_0);
plot(Table(:, 1), Table(:, 2))
xlabel('Excursion Time (seconds)')
ylabel('Energy (Joules)')
grid on
```

Note that the inputs used in the functions would be defined before and can be seen in the raw code in Appendix 2, at the top.

2.2 Theoretical Laser Profile

The code allows the user to specify a select number of equispaced atoms to place within the laser field. The laser profile is displayed as a Gaussian, which would take place at the focus of the IR laser. The profile is shown in figure 18.

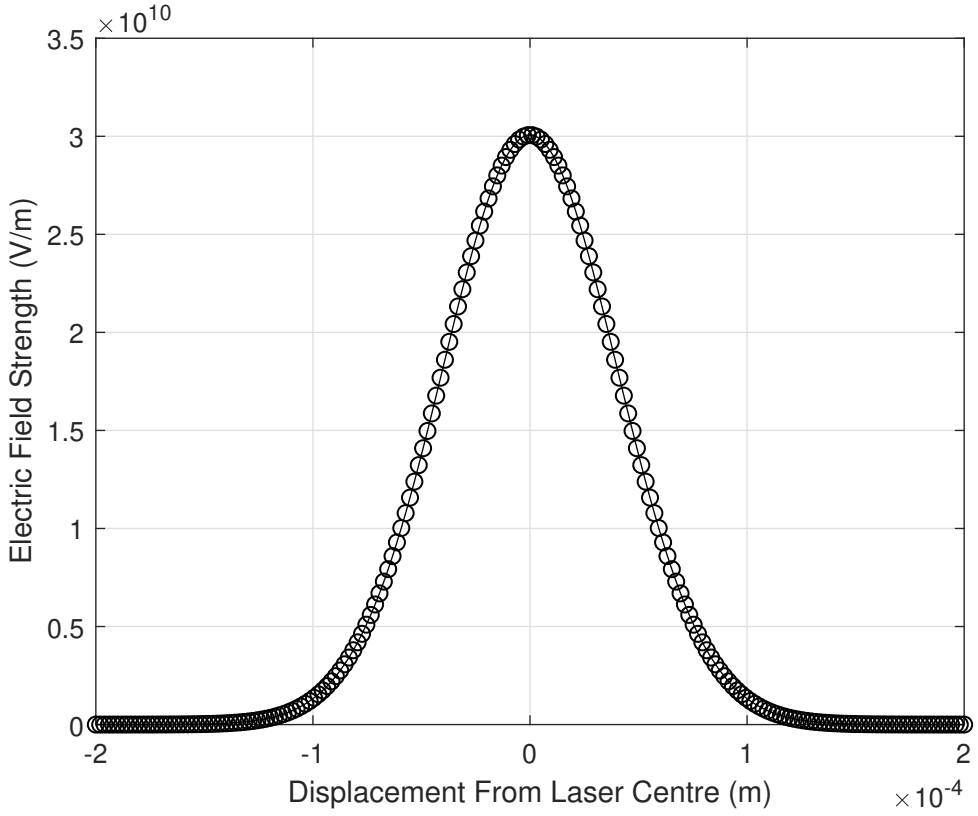


Figure 18: A visualisation of the laser field used for the results section. There are 200 atoms placed within a 200 micron radius laser profile. Each circle represents an atom, and its height represents the field that it experiences.

The default settings for the set-up are

```
Centre of interval = 0 m
Standard deviation = 0.2*(laser radius)
```

The electric field of an XUV photon signal before it is sent into the farfield is given by equation (19).

$$E_p \quad = \quad Ae^{i\theta} \tag{19}$$

where E_p is the electric field of the photon, A is the XUV intensity and θ is the phase of the XUV photon, given by the number trajectory it belongs to.

An important relationship to be made is the relationship between the intensity of the IR laser and the intensity of emitted XUV photons. The raw intensity of the produced XUV photons is usually on the order of about 10^6 lower than that of the IR laser, however this would imply a linear relationship between the two.

For the purpose of this thesis, the relationship is taken to be non-linear, with a relationship given by equation (20).

$$\text{Intensity}_{\text{xuv}} \quad = \quad (\text{Intensity}_{\text{laser}})^6 \tag{20}$$

This is done for shaping purposes, in order to account for the fact that a higher intensity will lead to a greater quantity of XUV photons being liberated from the target medium in a

non-linear manner.

The resultant number for XUV photon intensity, although fundamentally incorrect in terms of raw intensity value, will be normalised against the highest XUV photon intensity in the set it belongs to, meaning that the XUV photon intensity will be labelled "arbitrary".

3 Results

Results and outcomes of the code are put into this section. Results will be placed without explanation other than the figure description here – further discussion of them is reserved for section 4. If not otherwise specified, the settings used to produce results are, as default values:

```
Laser Wavelength: 800 nm
Laser Intensity: 1.2e18 W/m^2
Atom: Argon
```

3.1 The Third Trajectory

3.1.1 Single Atom - Phase Location

Expanding upon figure 7, figure 19 is presented, which demonstrates the position of the third trajectory according to this approximation of the HHG process.

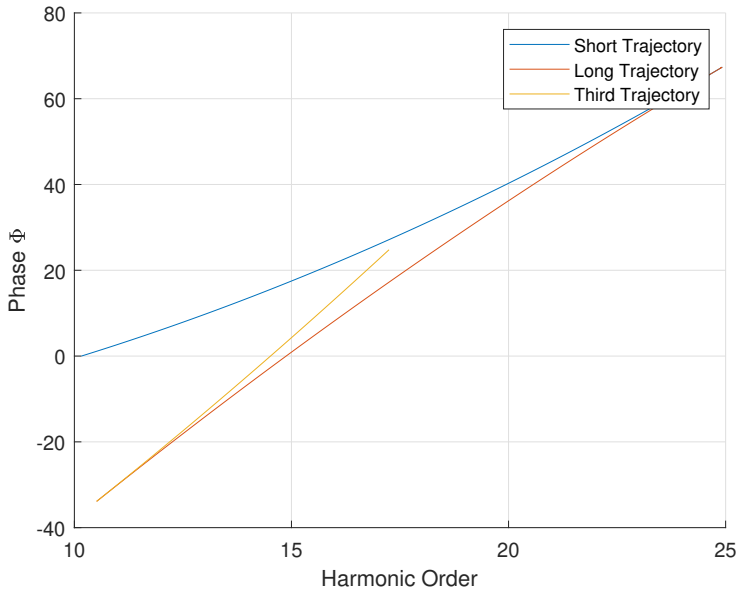


Figure 19: According to the approximation discussed in section 1.2.3, the phase of the third trajectory is given.

3.1.2 Unlikelihood of Observation/Electron 'Survival'

This simulation begins with 4097 electrons, all released at different points within a half-cycle period. This is based upon an 800 nm laser, operating at an intensity of 1.2e18 W/m^2. There are 4097 unique trajectories displayed and the graph shows how many are in effect "stable" enough to produce further trajectories.

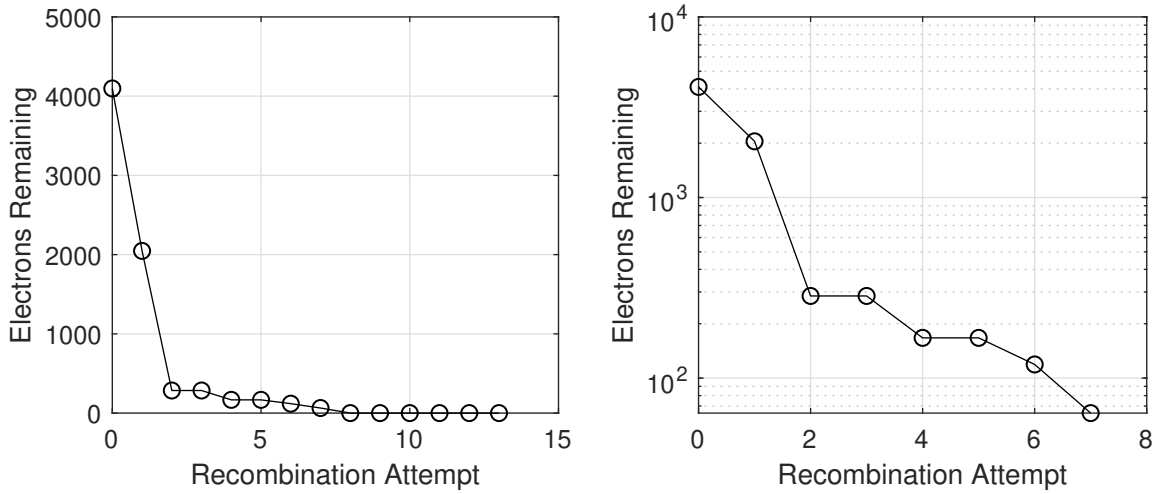


Figure 20: Both graphs correspond to the "survival rate" of an electron after a number of cycles. Left: linear y axis, right: log y axis.

3.1.3 Many Atoms, Far-field Projection

Figure 21 shows an example of the third trajectory in the far-field.

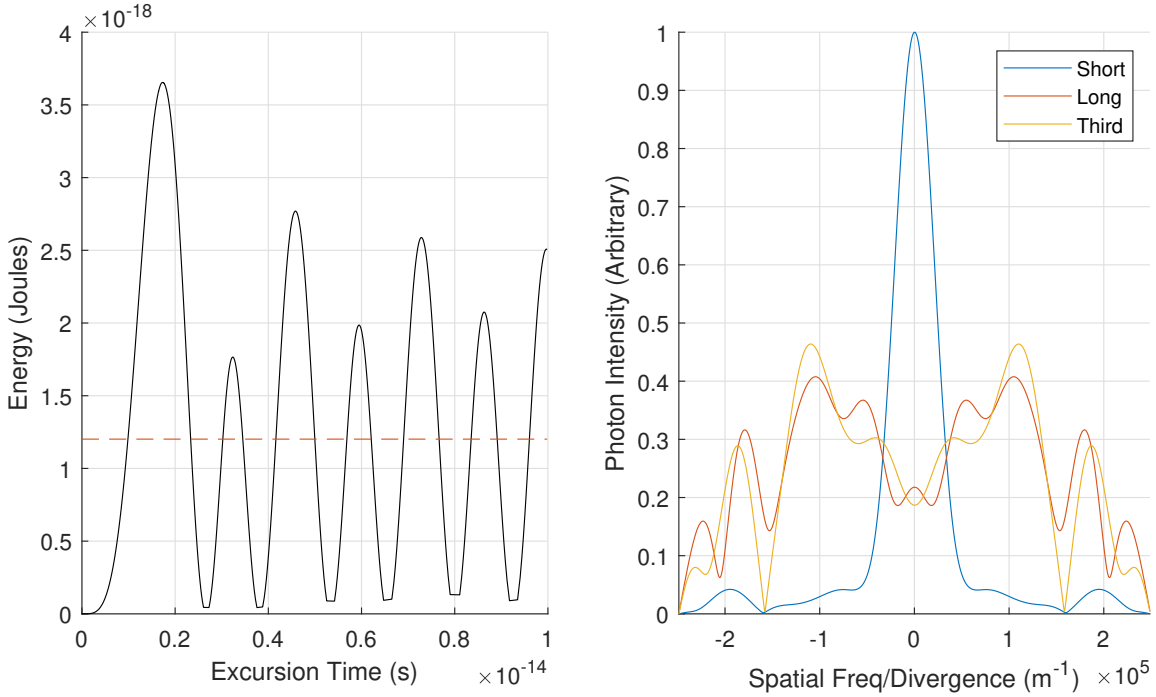


Figure 21: Left: The fifteenth harmonic (red dotted line) on the laser's respective energy/time graph. Right: The fifteenth harmonic projected in the far-field, with first, second and third trajectories visible.

3.2 Further Trajectories

3.2.1 Single Atom - Phase Locations

If the code is used to find phases up to the tenth trajectory, figure 22 results.

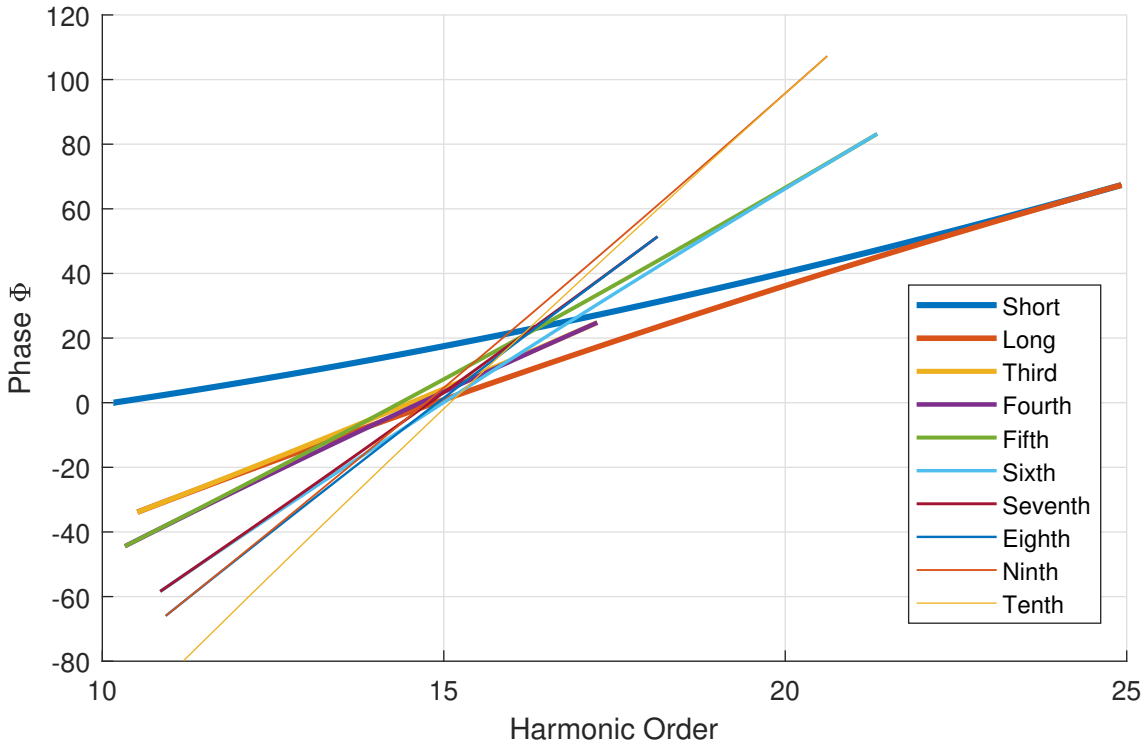


Figure 22: According to the approximation discussed in section 1.2.3, the phase of the further trajectories is given.

3.2.2 Far-field Projection

The harmonic cut-offs can be seen in figure 22, resulting in a far-field diagram of 23.

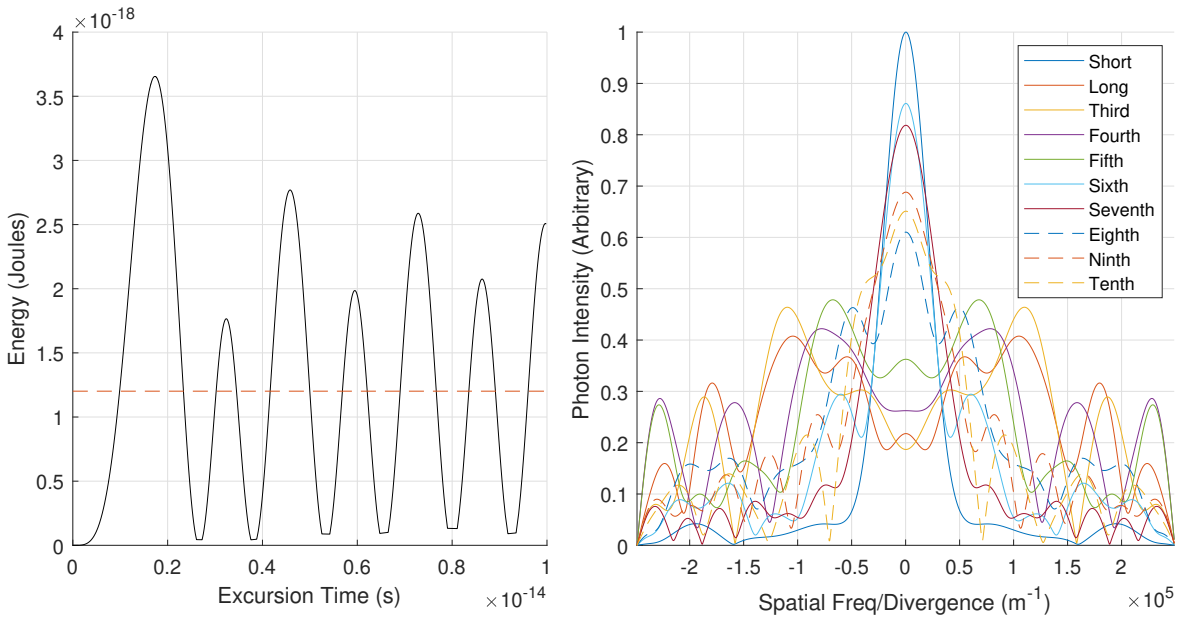


Figure 23: The fifteenth harmonic projected in the far-field, with trajectories up to the tenth trajectory visible.

3.3 Far-field Projections of Different Harmonics, Third Trajectory

The following settings are still in use through these results;

```
Wavelength: 800nm
Atom: Argon
```

The laser intensity will be given in the subsection header, and the harmonics will be given in the figure description. In order to demonstrate where the harmonics lie, an energy spectrum is provided prior to each figure.

3.3.1 $1.0 \cdot 10^{18}$ W/m^2 Intensity

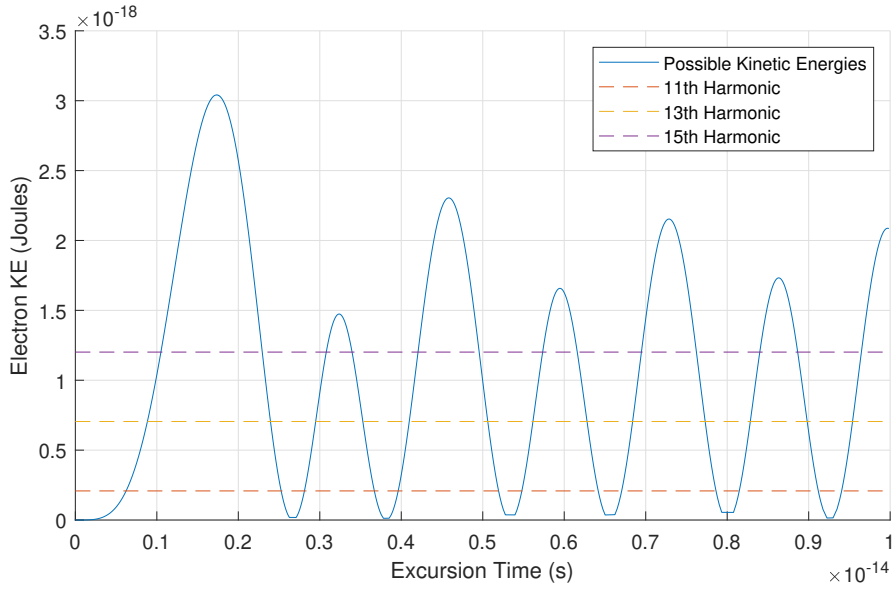


Figure 24: Odd harmonics capable of displaying a third trajectory for a laser source with intensity 1.0e18 W/m^2.

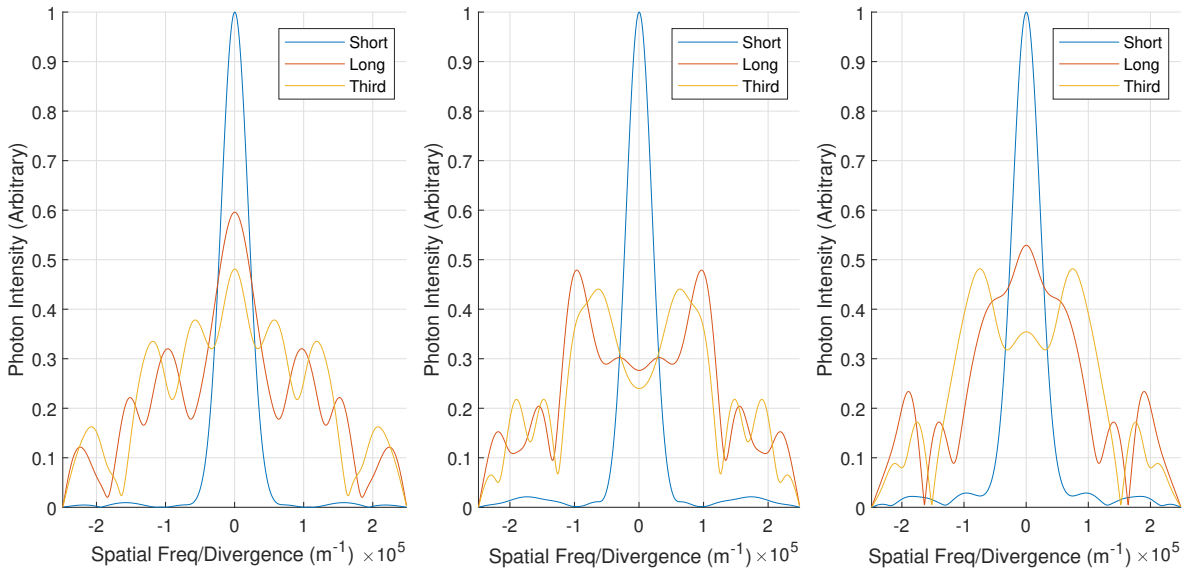


Figure 25: 1.0e18 W/m^2 laser source intensity, far-field image. Left: 11th harmonic, centre: 13th harmonic, right: 15th harmonic.

3.3.2 1.2 $\cdot 10^{18}$ W/m^2 Intensity

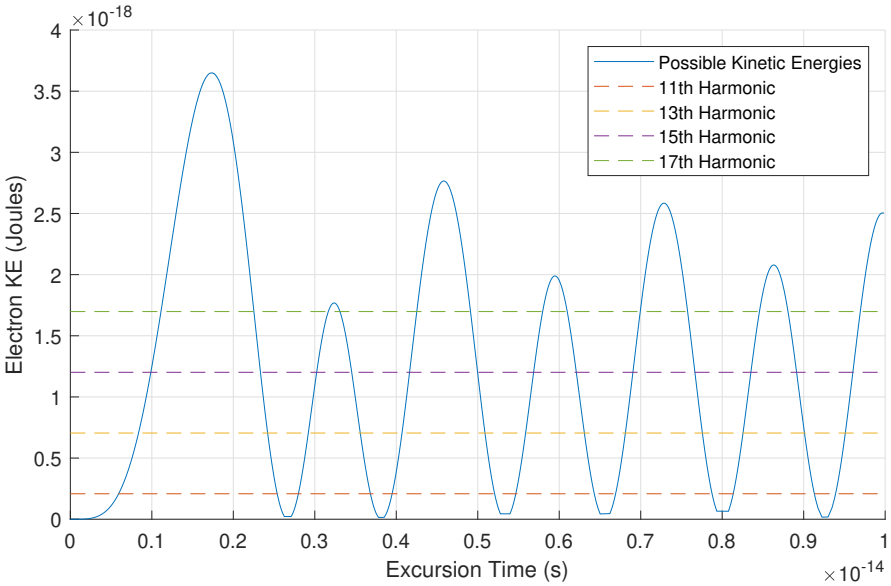


Figure 26: Odd harmonics capable of displaying a third trajectory for a laser source with intensity 1.2e18 W/m^2.

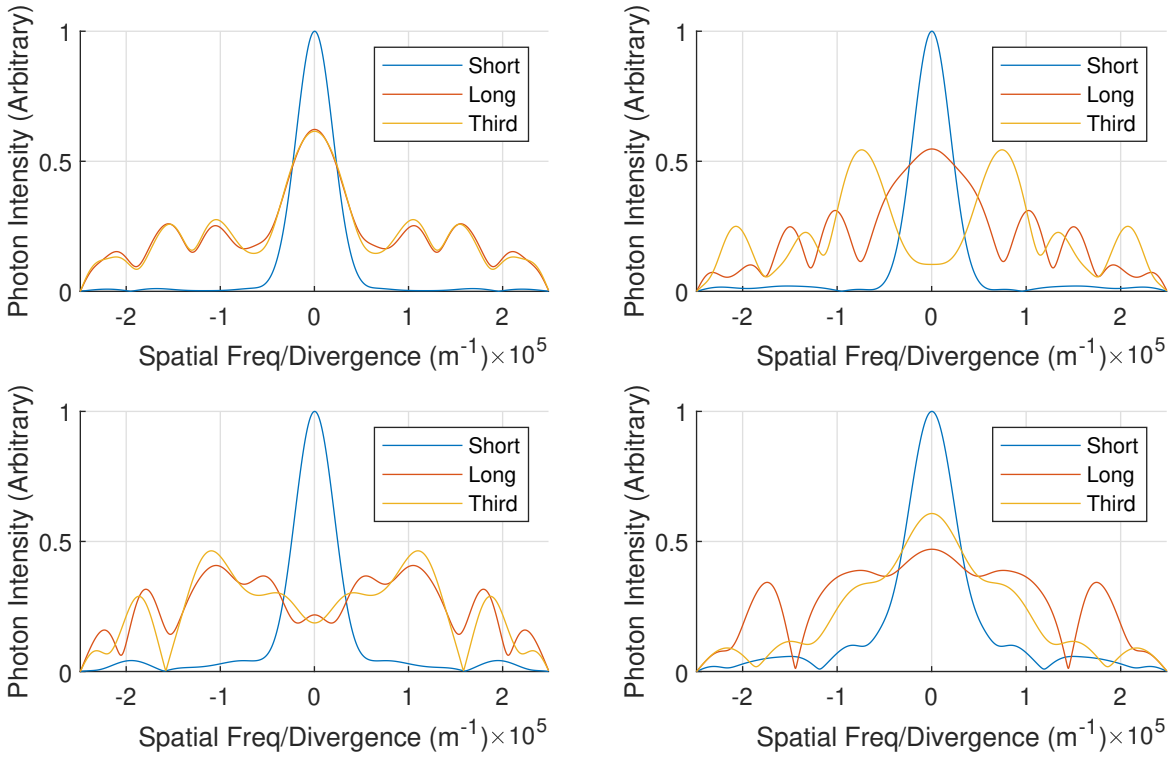


Figure 27: 1.2e18 W/m^2 laser source intensity, far-field image. Top left: 11th harmonic, top right: 13th harmonic, bottom left: 15th harmonic, bottom right: 17th harmonic.

3.3.3 2.0 $\cdot 10^{18}$ W/m^2 Intensity

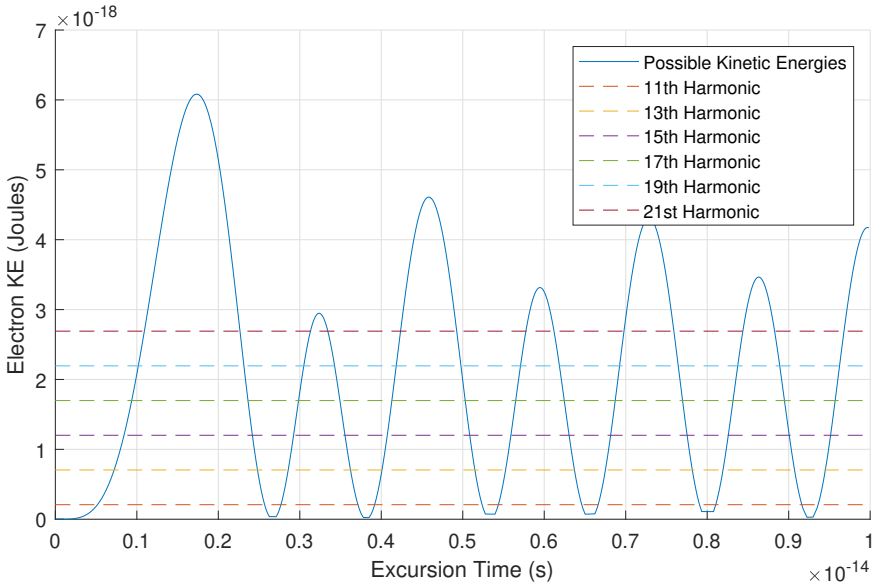


Figure 28: Odd harmonics capable of displaying a third trajectory for a laser source with intensity 2.0e18 W/m^2.

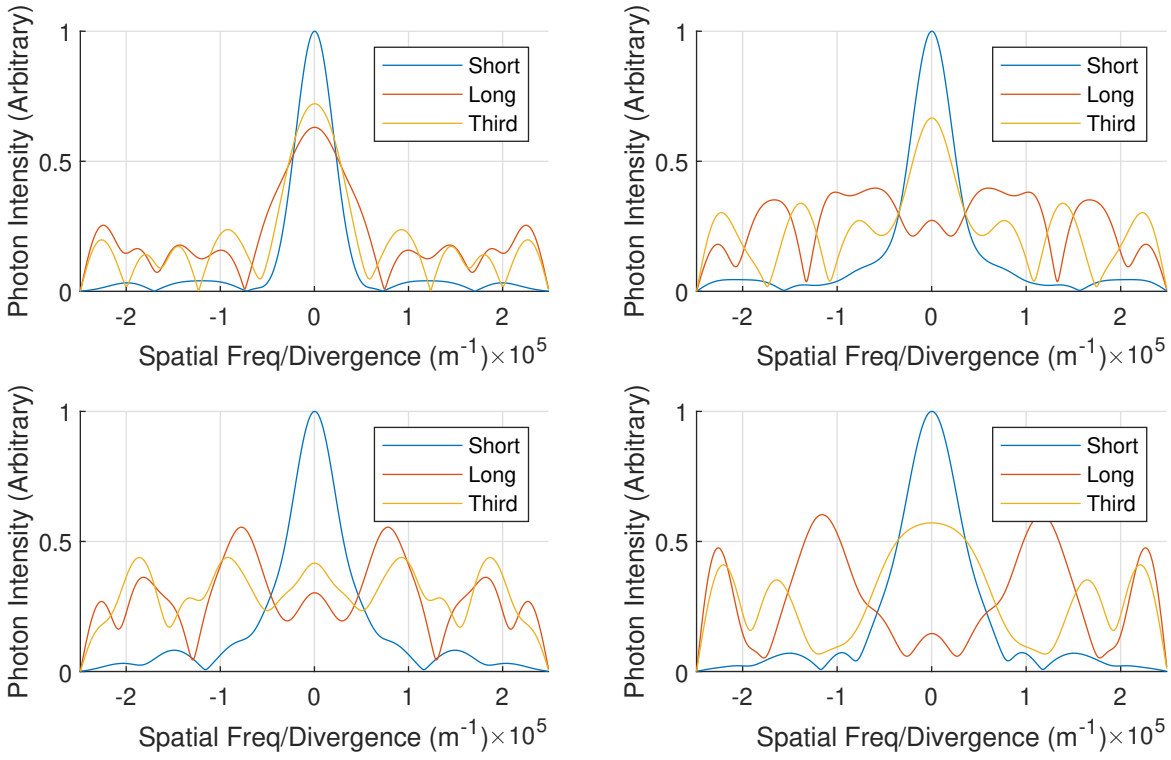


Figure 29: 2.0e18 W/m^2 laser source intensity, far-field image. Top left: 15th harmonic, top right: 17th harmonic, bottom left: 19th harmonic, bottom right: 21st harmonic.

4 Discussion

As far as the author is aware at this present moment from reading the literature, the third trajectory has only been theoretically predicted, but not yet experimentally observed [5, 3, 9]. There are a great number of difficulties in performing this task, but in an attempt to display one or two of them, figure 20 has been plotted.

The "survival rate" of different trajectories has been measured, by noting the number of electrons that still have the possibility to recombine after each pass of the atom. This figure in-effect measures the number of paths remaining which do not diverge due to the electric field.

Figure 20 was plotted based upon 4097 trajectories, which had ejection times $0 \leq t \leq 0.5$ IR cycles (as seen in figure 4).

- Half of the electrons do not get a chance to recombine, due to being ejected by the unfavourable phase of the electric field.
- By the second opportunity to recombine (one pass of the atom, trajectories 3-4), less than ~7% of the initial electron population remains.
- By the 8th pass of the atom, less than 1% of the initially ejected electrons can still recombine.

The third point in this list is not useful in any manner besides to demonstrate that the chance of an electron ejecting at such a favourable time is unlikely.

A second reason why the third trajectory might be difficult to locate experimentally is that in terms of phase (if based upon the SFA), it appears to fall between the first and second trajectories. It becomes apparent that a good place to search for the third trajectory might be where its separation is maximised from the short and long trajectories simultaneously. Looking at figure 19, which is for an 800 nm laser at a 1.2e18 W/m^2 laser intensity, this would in theory be around harmonic 17 for Argon. This is interestingly not supported by figure 27, which appears to suggest that in this set-up, the third trajectory seems to be 'obscured' by the other two trajectories. A possible reason for this though is perhaps that the further trajectories require more sampling than the shorter ones in order to get them to a high degree of fidelity – which might mean that the third trajectory is under-sampled here.

Although not taken into account in this semi-classical-based thesis, a third reason for the difficulty in finding the third harmonic is that the more accurate models regard the electron as a wave packet which spreads over time, which has a dependence upon the wavelength of the driving laser [14].

Based upon figures 26 and 27, a method of seeing if this is possible is perhaps by process of elimination, using cut-off energies. In figure 26, the 17th harmonic sits nearly at the peak of the third trajectory – therefore observing the energies of the 15th and 19th harmonic would give a 'control' on either side. Ideally, the 19th harmonic would be a signal which has less 'noise' (due to the fact that the fact that this energy is beyond energies that the third trajectory is capable of producing).

It is possible to upon looking at figure 22 to consider that viewing the 17th harmonic might present its own problems, namely that there appear to be a large cluster of trajectories in this region and hence there might be a reasonably large amount of 'noise', although the 'noise' appears to peak around harmonic 15. This worry can however be remedied based upon the qualitative discussion of statistics previously had on "surviving electrons" earlier. In other words, with every trajectory, the chance of observing it becomes notably less as the trajectory number increases, purely by dint of the few electrons that are not expelled by an unfavourable oscillation in the laser field.

There is an aspect of the code that has been artificially chosen in this thesis instead of being programmed, which is the interplay between the harmonics. In a real experiment, one would witness a dampening of the even harmonics, which can be seen in figure 2.5c of Neven's thesis [3]. The reason for this dampening of the even harmonics is due to the half-cycle rate, as oppose to a single-cycle rate. In the code, this is neglected, as only electrons born from one half-cycle of the IR are simulated – however it is accounted for by only selecting odd harmonics.

The top-right image in figure 27 shows the second and third trajectories behaving differently (one peaks when the other troughs) for the 13th harmonic, which is surprising as the second and third trajectories only have a very slight difference in phase at this value.

One potential explanation for this might be an unintentional "single slit" effect exhibited by the third trajectory. Figure 18 shows that the beam is narrow, meaning that the intensity fades very quickly as one moves away from the centre. The reality of this is that the third trajectory will stop being produced at a much lower cutoff than the first and second trajectories. Although this is a concern – it should reflect an experimental situation. The third trajectory cannot be produced under its respective cutoff energy. This allows for another interesting reason as to why detection of the third trajectory is difficult – namely because the pattern for a single-slit is very dense spots of light, while as many-slit tend to exhibit greater distances between the nodes. It's possible that the central node from the first and second trajectories are masking the third. As far as can be seen, the only way around this would be to control a laser pulse in such a way where the beam is "square" in nature – ie. all points of equal intensity on the wave front.

The intensities used in the results section were chosen to give a range to choose from when looking at results. It is clear from figure 24 that a lower intensity laser produces a lower number of harmonics, which is to be expected. This can be replicated similarly by using a lower frequency laser. For example, if one were to use a 1000 nm laser instead of an 800 nm one, more harmonics could be observed.

As well as there being a cut-off for the highest harmonic that a particular set-up (laser wavelength, laser intensity and medium composition) can achieve, there is also a cut-off of minimal energy as well, due to the energy gained by the photon upon the electron re-combining with the atom.

This thesis has operated on a semi-classical framework and does not take into account the more recent developments and theories of HHG. While limited to this semi-classical framework, the best that one can do is show how one might go about predicting where one might find the third trajectory through use of the model.

As expressed in the paragraphs above, the search for the third trajectory (if the goal of an experiment) might be best suited in simply probing around the cut-off value of the third trajectory for that particular laser.

5 Acknowledgements

As a result of the COVID-19 pandemic - I have been deprived the pleasure of getting to know many people that I would have otherwise known if this were a "normal" bachelor process. Even though the process has been all online, I still owe a debt of gratitude to many.

Firstly, I would like to thank Isa Hendriks. If we had not had that small discussion in the hallway about your bachelor degree, I would never have discovered this interesting topic and the possibility of working under Johan.

I would like to thank my two supervisors Johan Mauritsson and Samuel Bengtsson. Johan, for taking me on in the first place, your tutorial on the Fourier transform in Swedish, and guidance when requested. I am also very grateful that you listened to what I wanted to learn during my bachelor, and picked the perfect topic – I have learned a lot from this. Samuel, for answering my constant stream of questions with not only patience, but taking the time to contact other people and get their inputs as well. Your several readings and suggestions for additions/changes to my thesis have been greatly valuable as well.

My sambo Emma, who is the light of my life and has been for the last 10 years. You inspire me with your compassion and hard work with your nursing degree and I believe you will be a fantastic nurse.

I would also like to thank Ivan Ahmed – who I have almost had a symbiosis with, in the sense that I've complained about my bugs in my code and numerous other small things that I've been stuck on, and you've likewise ranted about particular difficulties in your medical science course. You have certainly been an agent of sanity for me – thank you.

6 Appendices

6.1 Appendix 1 - Fourier Transform Steps

In order to perform a Discrete Fourier transform, the following steps are taken:

1. A number of samples is chosen, N.
2. Two times are chosen as the initial and final sampling times, between which the N samples are evenly distributed. The sampling rate is calculated as being the equal to the difference between the initial and final time, divided by the number of samples. The sampling frequency is then the reciprocal of the sampling rate.
3. The amplitude of the wave at each of these sampling times is noted, wherein x_0 will correspond to the amplitude at the 'zeroth' sampling time, and so on.
4. Once a vector is obtained of amplitudes at all sampling times, one can begin to calculate the frequency samples using equation (14). (For example: For X_0, set $k = 0$ as a constant, and then use all of the amplitude samples to complete.)
5. The magnitudes of the X_k values are taken, and the x axis is then scaled to correspond to frequency, rather than arbitrary *bins*. This is done as follows:
   (a) A number line is set up, which extends from 0 to $N - 1$, in steps of 1.
   (b) This line is then multiplied by the sampling resolution, which the inverse of the total sampling time.
6. The resultant plot can be symmetrical, which is an artefact of the trigonometry and how it is represented in terms of Euler's formula and mentioned at the end of section 1.3.1. Provided that the sampling frequency is high enough, the amplitudes to the left of the Nyquist Limit are doubled and then divided by N.

6.2 Appendix 2 - Code Functions

This section offers an explanation of all of the functions, and their purpose. For the purpose of not being repetitive, if an input has been explained in another function, it will not be explained again.

Variable names will be put in **bold**, with their explanation following them. In all of the functions where a table is produced, there is a comment provided just below the function definition which explains what the different columns are.

VelocityCalculator/DisplacementCalculator

Inputs:

t_i/initialTime, the ionisation time of the electron. **Time**, a list containing all moments of time between two set boundaries. **q**, the electron charge constant. **E0**, the electric field strength. **m_e**, the mass of an electron. **w_0**, the angular frequency of the driving laser. **returnTime**, the recombination time of the electron with the atom.

Description:

These functions use equations (3) and (4) to find the velocity and displacement values at a particular instant of time. Velocity is never used for presentation, but it is used in the calculation of energy.

DisplacementMatrixMaker

Inputs:

EjectionTimes, a list of ionisation times between two periods, equidistant.

Description:

This function creates a matrix containing where the electron is at all times, for the different ionisation times presented to it. The first column represents all of the times within the simulation, and all columns after it represent different ionisation times. One can find out what the ionisation times are by finding the first instant where the displacement has a non-zero value.

BasicEnergyTableMaker

Inputs:

DisplacementMatrix, which is the output of DisplacementMatrixMaker. **lastEjectionTime**, which is the last possible time that an atom can be ionised, chosen by the user.

Description:

This function creates a table which houses a lot of information about the different trajectories that are possible with the starting conditions provided. The initial purpose of this function was to produce an energy/time plot to show the ponderomotive energy, and hence "PonderoTable" is another name given.

EnergyTableExtended

Inputs:

PonderoTable, which is the output of BasicEnergyTableMaker.

Description:

This function's purpose is to add another two columns, one where the recombination time is fit with linear functions in an attempt to simplify the shape, and another which uses a definition proposed by Chen Guo in their thesis to define phase. This is discussed further in section 1.2.

ColourTableCreator & ColourMapUser

Inputs:

IncrementCount is how many different shades the user would prefer. **RGBLow & RGB-High** are the RGB value of the lowest and highest value colours respectively. **ColourMap** is the output of ColourTableCreator. **Value** is the numerical value which needs to be assigned a colour. **MinValue & MaxValue** are values designated with RGBLow and RGBHigh to create boundaries.

Description:

With how DisplacementMatrixMaker was programmed, it houses within it all of the information about the different trajectories. Due to the fact that an electron can fail to recombine with the atom on a pass, there is a need to re-calculate the energy of the trajectory based upon the next recombination time. This means that the already-defined functions in MATLAB became inconvenient to use. These functions effectively allow the user to choose what the colours are when viewing a trajectory plot.

RecombinationFinder

Inputs:

matrixTimeFirstCol is the output of DisplacementMatrixMaker, where there was an emphasis on the fact that the first column must be 'Time'.

Description:

This function takes the DisplacementMatrix output and finds all of the recombinations and "passes" of the electron. There are many different ways to program this, but the method that was used in the end is to check for a change in sign. This function would be redundant if the trajectories were all continuous, or if the displacement equation had an analytical solution. Unfortunately the points are discrete, meaning that the solutions are not exact in this program – they are approximations.

IntensityToElectricFieldCalculator

Inputs:

I_in_WattsperSquaredMetre, **vacuumPermitivity_inFaradsperMetre** and **speedOfLight_inMetr**
which are self-explanatory. The reason for the long variable names is that it is important to ensure that units are clear.

Description:

The purpose of this function is to allow an easy conversion between a laser intensity, and the maximum electric field strength. Useful because electric field strength is usually the variable used in equations, but intensity is often the supplied quantity.

TrajectoryPlotter

Inputs:

CMap is the output of ColourTableCreator.

Description:

The purpose of this function is to take in the variables provided and produce a graph showing all of the trajectories, which are all coloured with their respective energies. Light grey indicates a trajectory that diverged without a single chance to recombine, while as darker grey indicates a trajectory that has passed the atom at least once.

EnergyExaminer

Inputs:

energyOfInterest is the electron kinetic energy, in Joules, that the user wants to test, **E0_1_by_n_List** is for the case that the user wants to provide a list of electric field values to run through the examiner (in other words, if the laser's electric field changes), **IncludeFurtherTrajectoriesYN** is answered with "Y" or "N", which will include/exclude trajectories after the long trajectory.

Description:

The purpose of this function is to allow the user to get a lot of information about one particular electron kinetic energy. One can check a photon energy as well by removing the ionisation potential from it.

LaserSimulator

Inputs:

maxDisplacement is the radius of the laser which is shone on a sample size. For example, 200e-6 will simulate a laser shining on a circle of 200 micron radius. **numberOfAtoms** is self-explanitory. **maxE0** is the maximum electric field of which the laser is capable. By default, the laser simulator assumes a 1D Gaussian shape, with a pre-set standard deviation. The user can adjust this for their own purposes.

Description:

In one dimension, this function allows the user to simulate what happens to different atoms at different positions within the laser field. All of the energetically possible trajectories are recorded and can be accessed.

NearfieldEnvelopeBuilder

Description:

The purpose of this function is to produce a complex amplitude corresponding to the photon immediately after conception. This functions essentially only exists as an input to the next function, unless one wishes to see the real part of it.

FarfieldEnvelopeBuilder

Description:

This function takes the output of NearfieldEnvelopeBuilder and Fourier transforms it, moving from the spatial domain to the spatial frequency domain. The principle here is that, in the far-field, certain elements of the spatial domain will be represented differently (discussed in section 1.3.4). This function serves as destination of this thesis, which is to attempt to get the third trajectory to 'stand out'.

6.3 Appendix 3 - Code

6.3.1 Superposition Code

```
Time = [0:0.01:4*pi];

% First, plot the static waves

subplot(1, 3, 1)
BigWave1 = 0*Time -15;
hold on
for frequency = 1:1:5
    plot(Time, waveMaker(Time, frequency, 0, 2*frequency), 'Color', '[0 0 0 1]')
    BigWave1 = BigWave1 + waveMaker(Time, frequency, 0, 2*frequency);
end
plot(Time, BigWave1, 'Color', '[0 0 0 1]', "LineWidth", 1.33)
hold off
title('No Delay')
xlabel('Time (Arbitrary)')
ylabel('Intensity (Arbitrary Units)')
xlim([0 4*pi])

% Now, plot the waves with a LINEAR phase delay applied

subplot(1, 3, 2)
BigWave2 = 0*Time -15;
hold on
for frequency = 1:1:5
    plot(Time, waveMaker(Time, frequency, lin_phaser(frequency), 2*frequency), 'Color
        ', '[0 0 0 1]')
    BigWave2 = BigWave2 + waveMaker(Time, frequency, lin_phaser(frequency), 2*
        frequency);
end
plot(Time, BigWave2, 'Color', '[0 0 0 1]', "LineWidth", 1.33)
hold off
title('\xi(\omega) = Constant')
set(gca,'YTickLabel',[]);
xlabel('Time (Arbitrary)')
xlim([0 4*pi])

% Now, plot the waves with an EXPONENTIAL phase delay applied

subplot(1, 3, 3)
BigWave3 = 0*Time -15;
hold on
```

```
40 for frequency = 1:1:5
41     plot(Time, waveMaker(Time, frequency, exp_phaser(frequency), 2*frequency), 'Color
           ', '[0 0 0 1]')
42     BigWave3 = BigWave3 + waveMaker(Time, frequency, exp_phaser(frequency), 2*
           frequency);
43 end
44 plot(Time, BigWave3, 'Color', '[0 0 0 1]', "LineWidth", 1.33)
45 hold off
46 title('\xi(\omega) = Variable')
47 set(gca,'YTickLabel',[]);
48 xlabel('Time (Arbitrary)')
49 xlim([0 4*pi])
50
51 % Functions must go at the end of the file
52
53 function wavesum = waveMaker(Time, f, phi, adjustment)
54     wavesum = cos(Time*f + phi) + adjustment;
55 end
56
57 function lin_phase = lin_phaser(frequency)
58     lin_phase = frequency*2;
59 end
60
61 function exp_phase = exp_phaser(frequency)
62     exp_phase = exp(2*frequency+3).^2;
63 end
```

6.3.2 Discrete Fourier Transform Code

```
1 % Spatial Fourier Transform Code
2
3 subplot(1,2,1)
4
5 final = 10e-2;
6 initial = 0;
7 increment = (final-initial)/(1e4);
8 x = initial:increment:final; % seconds
9
10 w = 2*pi;
11 sum = x*0;
12
13 halfWay = length(x)/2;
14 broadness = floor(0.01*length(x));
15
16 sum(halfWay-broadness:halfWay+broadness) = 1;
17
18 % Write Fourier code to figure out frequencies
19
20 N = 1000;
21 samplingRate = (final-initial)/(N);
22 samplingFrequency = 1/samplingRate;
23 fprintf('The Nyquist limit is %f.\n', samplingFrequency/2);
24 fprintf('In order to get a good resolution, N has to be at least %f. \n', (final-
       initial)*2*maxF);
25 fprintf('Aperture size is %f. \n', (length(sum(halfWay-broadness:halfWay+broadness))*
       increment));
26
27 x_n = zeros([1, N]);
28 t_n = 0*x_n;
```

```
modifier = length(x)/N;

for n = 1:1:N
    t_n(1, n) = x(1, floor(modifier)*n);
    x_n(1, n) = sum(1, floor(modifier)*n);
end

hold on
figure(1)
set(gcf,'Position',[250 500 600 300])
plot(x, sum, "Color", "k")
%plot(t_n, x_n, "LineStyle", "--", "Marker", "o", "Color", "r") % This line shows the
    sampling
xlabel('Displacement (m)')
ylabel('Nearfield Intensity (Arbitrary Units)')
grid on

X_k = 0*x_n;

for k = 0:1:(length(X_k)-1)
    storageValue = 0;
    for n = 0:1:(length(x_n)-1)
        storageValue = storageValue + x_n(n+1)*exp((-1i*2*pi*k*n)/N);
    end
    X_k(k+1) = storageValue;
end

X_k = abs(X_k);

samplingResolution = 1/(final-initial);

xAxis = 0*x_n;

for index = 0:1:N-1
    xAxis(1, index+1) = index*samplingResolution;
end

halfWay = floor(length(xAxis)/2);
length(xAxis);

subplot(1,2,2)
set(gcf,'Position',[900 500 900 300])
yIntensity = fftshift(2*X_k/N);
plot(xAxis, yIntensity/max(yIntensity))
xlabel("Spatial Frequency/Divergence (m^{-1})")
ylabel("Farfield Intensity (Arbitrary Units)")
grid on
```

6.3.3 Full Simulation Code

```
% Tom Causer - Bachelor Project Code

% ....................................................................

% Constants Section

m_e = 9.11e-31; % kg
q = -1.602e-19; % Coulombs
c = 2.998e8; % m/s
```

```
10 vacuumPermitivity = 8.854e-12; % Farads/metre
11 h = 6.626e-34; % J*s
12
13 % ..................................................................
14
15 % Important user-chosen parameters
16
17 IR_Cycles = 4;
18 LaserWavelength = 800e-9; % meters
19 LaserIntensity = 1.2e18; % W/m^2
20 NoOfTimeIncrements = 2^13; % larger = more trajectories included, 2^13 default
21
22 % ..................................................................
23
24 f = c/LaserWavelength; % s^-1 (Hz)
25 w_0 = 2*pi*f; % laser angular frequency
26 lastEjectionTime = 0.5*(1/f); % last time that electrons can be ejected
27 TimeBegin = 0; % units: depends on frequency
28 TimeEnd = IR_Cycles*1/(f); % end time of simulation
29 Increment = (TimeEnd-TimeBegin)/NoOfTimeIncrements;
30 Time = TimeBegin:Increment:TimeEnd; % Column 1 in table
31 EjectionTimes = 0:Increment:lastEjectionTime;
32
33 % Energy related calculations
34
35 E0 = IntensityToElectricFieldCalculator(LaserIntensity, vacuumPermitivity, c); %
       Volts per metre
36 U_p = ((q^2)*(E0^2))/(4*(m_e)*(w_0^2)); % ponderomotive energy
37 maxEnergy = 3.18*U_p; % Set to 3.18 to prevent indexing errors
38 CMap = ColourTableCreator(NoOfTimeIncrements, [255, 0, 0], [255, 255, 0]);
39
40 % Chosen Harmonic
41
42 ionisation_potential = 15.75*abs(q); % 15.75 eV is Argon
43
44 SoughtAfterHarmonic = 15;
45 harmonicFrequency = f*SoughtAfterHarmonic; % Hz
46 photon_energy = h*harmonicFrequency; % Joules
47 electron_KE = photon_energy - ionisation_potential;
48 cutoff = ((3.17*U_p + ionisation_potential)/h)/(f);
49
50 % All functions
51
52 function FarfieldTable = FarfieldEnvelopeBuilder(maxDisplacement, numberOfAtoms,
       maxE0, Time, EjectionTimes, GroupOrPhaseDelay, energyOfInterest, q, m_e, w_0, c,
       vacuumPermitivity, h, IncludeFurtherTrajectoriesYN)
53     % The columns of this function are:
54     % (1) Farfield distance
55     % (2) Amplitude of the first trajectory
56     % (3) Amplitude of second trajectory
57     % (4+) and so on...
58
59     % First, "import" the nearfield table
60     nearFieldTable = NearfieldEnvelopeBuilder(maxDisplacement, numberOfAtoms, maxE0,
           Time, EjectionTimes, GroupOrPhaseDelay, energyOfInterest, q, m_e, w_0, c,
           vacuumPermitivity, h, IncludeFurtherTrajectoriesYN);
61
62     sampleSizeFactor = 5;
63     % Initialise a farfield table and take the first column as an x-axis
64     farFieldTable = zeros([length(nearFieldTable(:, 1))*sampleSizeFactor-(
```

```
        sampleSizeFactor-1), length(nearFieldTable(1, :))]);
65      DistanceList = nearFieldTable(:, 1);
66      farFieldTable(:, 1) = (0 : (1/(length(DistanceList)-1))/sampleSizeFactor : 1)/(
            DistanceList(2) - DistanceList(1));
67
68      % For each column in the nearfield table, go ahead and Fourier transform
69      % it using the fft function
70      for column = 2:1:length(farFieldTable(1, :))
71          farFieldTable(:, column) = fftshift(abs(fft(nearFieldTable(:, column), length
                (farFieldTable(:, 1)*sampleSizeFactor))));
72      end
73
74      % The aim with this next bit of code is to normalise the values
75
76      columnLength = length(farFieldTable(1, :));
77
78      % If you want to only do up to the third trajectory, have these next 3
79      % lines uncommented, otherwise comment them out.
80
81      if columnLength > 4
82          columnLength = 4;
83      end
84
85      % Find the max value in order to normalise all of the values against
86      % the highest value.
87
88      maxValue = max(max(farFieldTable(:, 2:columnLength)));
89      farFieldTable(:, 2:length(farFieldTable(1, :))) = farFieldTable(:, 2:length(
            farFieldTable(1, :)))/maxValue;
90
91      % Because of the fftshift function used above, the entire x axis needs
92      % to be translated half its maximum to the left.
93
94      farFieldTable(:, 1) = farFieldTable(:, 1) - 0.5*farFieldTable(end, 1);
95
96      FarfieldTable = farFieldTable;
97  end
98
99  function NearFieldTable = NearfieldEnvelopeBuilder(maxDisplacement, numberOfAtoms,
        maxE0, Time, EjectionTimes, GroupOrPhaseDelay, energyOfInterest, q, m_e, w_0, c,
        vacuumPermitivity, h, IncludeFurtherTrajectoriesYN)
100     % Outputs a table
101     %
102     % Columns are:
103     % (1) Displacement from laser centre
104     % (2+) Complex photon electric field values of increasing trajectories
105
106     atomTable = LaserSimulator(maxDisplacement, numberOfAtoms, maxE0, Time,
            EjectionTimes, GroupOrPhaseDelay, energyOfInterest, q, m_e, w_0, c,
            vacuumPermitivity, h, IncludeFurtherTrajectoriesYN);
107
108     % The next line solves an issue where values that go to NaN do not show
109     % up on the nearfield table even when they should.
110
111     %atomTable(isnan(atomTable)) = 0;
112
113     numRows = length(atomTable(:, 1));
114     numCols = length(atomTable(1, :));
115
116     % When building the template table, I remove two columns because I'm
```

```
117     % not going to use column 2 of atomTable, and columns 3 and onwards
118     % will be used to create an extra column called "Envelope"
119
120     outputTable = zeros([numRows, numCols-2]);
121     outputTable(:, 1) = atomTable(:, 1); % First column is displacement
122
123     % Go through the columns and fill in what the electric field values are
124
125     for column = 4:1:numCols
126         outputTable(:, column-2) = atomTable(:, 3) .* exp(1i.*atomTable(:, column));
127     end
128
129     % You now have the electric fields of the photons -- convert to
130     % intensities.
131
132     outputTable(:, 2:length(outputTable(1, :))) = 0.5*c*vacuumPermitivity*(
            outputTable(:, 2:length(outputTable(1, :)))).^2;
133
134     outputTable(isnan(outputTable)) = 0;
135
136     NearFieldTable = outputTable;
137
138 end
139
140 function LaserTable = LaserSimulator(maxDisplacement, numberOfAtoms, maxE0, Time,
        EjectionTimes, GroupOrPhaseDelay, energyOfInterest, q, m_e, w_0, c,
        vacuumPermitivity, h, IncludeFurtherTrajectoriesYN)
141 % This function returns a table that contains the following columns
142 %
143 % (1) Displacement of atoms
144 % (2) The external electric field applied to each atom
145 % (3) The intensity of the photons produced from each atom
146 % (4+) The remaining columns represent the phases of further trajectories
147
148 intervalSize = maxDisplacement*2/(numberOfAtoms-1);
149 centreOfInterval = 0; % centre of laser
150 atomPlacement = -maxDisplacement:intervalSize:maxDisplacement; % laser diameter
151
152 %laserToPhotonIntensityReductionFactor = 10^(-6);
153 laserToPhotonIntensityShapingFactor = 6;
154
155 % Gaussian shape is a*exp(-(x-b)^2/(2c^2))
156 % where a=height of peak, b=centre of peak, c=standard deviation
157 gaussianStandardDeviation = maxDisplacement*0.2;
158 E0_function = maxE0*exp(-(atomPlacement-centreOfInterval).^2./(2*
        gaussianStandardDeviation^2));
159
160 phaseTable = EnergyExaminer(Time, EjectionTimes, GroupOrPhaseDelay, energyOfInterest,
         E0_function, q, m_e, w_0, c, vacuumPermitivity, h, IncludeFurtherTrajectoriesYN)
        ;
161
162 % I want to find the number of extra columns required due to there being
163 % multiple trajectories that give the energy.
164
165 additionalColumnCount = 0;
166 for E0_Index = 1:1:length(E0_function)
167     E0Value = E0_function(E0_Index);
168     numAppearances = sum(phaseTable(:, 4)==E0Value);
169     if numAppearances > additionalColumnCount
170         additionalColumnCount = numAppearances;
```

```
171     end
172 end
173
174 combinedTable = zeros([length(atomPlacement), 3 + additionalColumnCount]);
175
176 % These are the "predefined columns"
177 combinedTable(:, 1) = atomPlacement;
178 combinedTable(:, 2) = E0_function;
179 %combinedTable(:, 3) = 0.5*c*vacuumPermitivity*(E0_function.^2)*
        laserToPhotonIntensityReductionFactor;
180 combinedTable(:, 3) = 0.5*c*vacuumPermitivity*(E0_function.^2).^(
        laserToPhotonIntensityShapingFactor);
181
182 % There will then be more columns which will be filled up with the
183 % different phases
184
185 % Begin by looking at the different values in phaseTable
186
187 for phaseIndex = 1:1:length(phaseTable(:, 2))
188     currentPhaseValue = phaseTable(phaseIndex, 7);
189     respectiveE0Value = phaseTable(phaseIndex, 4);
190     respectiveTrajectoryNumber = phaseTable(phaseIndex, 5);
191
192     % First, find the equivalent E0 value in combinedTable
193     combinedTableRowIndex = find(combinedTable(:, 2) == respectiveE0Value);
194
195     % Fill in the phase value, using the trajectory number column
196     combinedTable(combinedTableRowIndex, 3+respectiveTrajectoryNumber) =
            currentPhaseValue; %#ok<FNDSB>
197 end
198
199 % Want to pontentially make the values of 0 in the phase columns change to "NaN" so
        that
200 % they don't show up on graphs.
201
202 phaseColumns = combinedTable(:, 4:3 + additionalColumnCount);
203 phaseColumns(phaseColumns==0) = nan;
204 combinedTable(:, 4: 3+additionalColumnCount) = phaseColumns;
205
206 % We want to get rid of useless columns which are just full of NaN
207
208 lastFilledColumn = 0;
209
210 for column = 1:1:length(combinedTable(1, :))
211     NanBoolean = isnan(combinedTable(:, column));
212     notNaN = find(NanBoolean==0); %#ok<EFIND>
213     if isempty(notNaN)
214         lastFilledColumn = column-1;
215         break
216     end
217 end
218
219 combinedTable(:, lastFilledColumn+1:length(combinedTable(1, :))) = [];
220
221 LaserTable = combinedTable;
222 end
223
224 function PhotonInformation = EnergyExaminer(Time, EjectionTimes, GroupOrPhaseDelay,
        energyOfInterest, E0_1_by_n_List, q, m_e, w_0, c, vacuumPermitivity, h,
        IncludeFurtherTrajectoriesYN)
```

```
225 % Designed to zone in on an energy of the user's choice, and find out
226 % various things about that particular energy.
227
228 % GroupOrPhaseDelay is "phase" or "Phase" to show phase, and "Group" or
229 % "group" to show group delay. The point of the E0_1_by_n_List is that it
230 % allows you to find energy values for different values of E0, when cycling
231 % through a list. Very useful when working together with the laser profile
232 % function.
233
234 % The columns are as follows
235
236 % (1) Laser intensity
237 % (2) Recombination Time
238 % (3) Excursion time
239 % (4) E0 Value
240 % (5) TrajectoryNumber
241 % (6) Photon Energy
242 % (7) Phase
243 %
244 % The table is, by default sorted by phase.
245
246 numberOfIRCycles = round(Time(end)/(2*pi/w_0));
247 outputMatrix = zeros(numberOfIRCycles*4, 7);
248
249 outputRowCounter = 1;
250
251 for EValue = E0_1_by_n_List
252     % First, take a particular E0 value through the cycle
253     Intensity = 0.5*c*vacuumPermitivity*(EValue^2);
254     Matrix = DisplacementMatrixMaker(Time, EjectionTimes, q, EValue, m_e, w_0);
255     lastEjectionTime = Matrix(length(Matrix(1, :)), 1);
256
257     energyIndices = zeros([1, length(numberOfIRCycles*4)]);
258     energyIndicesCounter = 1;
259
260     BasicPonderoTable = BasicEnergyTableMaker(Matrix, lastEjectionTime, q, EValue,
        m_e, w_0);
261     EValuePonderoTable = EnergyTableExtended(BasicPonderoTable, h, w_0);
262
263     % Iterate through all excursion times, find their associated energies,
264     % and note if the energy of interest is found.
265
266     for excursionValueIndex = 2:1:length(EValuePonderoTable(:, 1))
267         currentEValue = EValuePonderoTable(excursionValueIndex, 2);
268         previousEValue = EValuePonderoTable(excursionValueIndex-1, 2);
269         if energyOfInterest > previousEValue && energyOfInterest <= currentEValue
270             energyIndices(1, energyIndicesCounter) = excursionValueIndex;
271             energyIndicesCounter = energyIndicesCounter + 1;
272         elseif energyOfInterest < previousEValue && energyOfInterest >= currentEValue
273             energyIndices(1, energyIndicesCounter) = excursionValueIndex;
274             energyIndicesCounter = energyIndicesCounter + 1;
275         end
276     end
277
278     % Alter for the possibility that the desired energy might not be found
279     % at all, due to the laser intensity on the edges being too weak
280
281     if isempty(energyIndices) == 1 | energyIndices == 0 %#ok<OR2>
282         continue
283     end
```

```
284
285     % At this point, we have all of the points which "collide" with the
286     % energy of interest. Now we need intensity and phase. Intensity is the
287     % same for each value of E0, but the phase is different for all. This
288     % is solved by getting the recombination times from the third column of
289     % EValuePonderoTable and dividing by the cycle time, to get the phase.
290     % We then put this into a table.
291
292     if GroupOrPhaseDelay == "Phase" || GroupOrPhaseDelay == "phase"
293         phaseChoice = 7;
294     else
295         phaseChoice = 6;
296     end
297
298     for index = 1:1:length(energyIndices)
299         indexOfInterest = energyIndices(index);
300         outputMatrix(outputRowCounter, 7) = EValuePonderoTable(indexOfInterest,
                phaseChoice);
301         outputMatrix(outputRowCounter, 6) = EValuePonderoTable(indexOfInterest, 2); %
                Photon energy
302         outputMatrix(outputRowCounter, 5) = EValuePonderoTable(indexOfInterest, 4); %
                TrajectoryNumber
303         outputMatrix(outputRowCounter, 4) = EValue; %E0 Value
304         outputMatrix(outputRowCounter, 3) = EValuePonderoTable(indexOfInterest, 1); %
                Excursion Time
305         outputMatrix(outputRowCounter, 2) = EValuePonderoTable(indexOfInterest, 3); %
                Recombination time
306         outputMatrix(outputRowCounter, 1) = Intensity; % Laser Intensity
307         outputRowCounter = outputRowCounter + 1;
308     end
309 end
310
311 % Now, get rid of any rows that take into account trajectories further than
312 % the ones we're interested in (allow user to choose maximum excursion
313 % time)
314
315 timePeriod = 1/(w_0/(2*pi));
316
317 if IncludeFurtherTrajectoriesYN == "n" || IncludeFurtherTrajectoriesYN == "N"
318     rebuiltMatrix = 0*outputMatrix;
319     rMatrixRowCounter = 1;
320
321     for row = 1:1:length(outputMatrix(:, 1))
322         if outputMatrix(row, 3) < timePeriod
323             rebuiltMatrix(rMatrixRowCounter, :) = outputMatrix(row, :);
324             rMatrixRowCounter = rMatrixRowCounter + 1;
325         end
326     end
327 else
328     rebuiltMatrix = outputMatrix;
329 end
330
331 sortedRebuiltMatrix = sortrows(rebuiltMatrix, 2);
332 nonZeroIndex = find(sortedRebuiltMatrix(:, 2) > 0);
333 sortedRebuiltMatrix = sortedRebuiltMatrix(nonZeroIndex(1):nonZeroIndex(end), :);
334 %sortedRebuiltMatrix = sortrows(sortedRebuiltMatrix, 4);
335
336 PhotonInformation = sortedRebuiltMatrix;
337 end
338
```

```
339 function BasicEnergyTable = BasicEnergyTableMaker(DisplacementMatrix,
        lastEjectionTime, q, E0, m_e, w_0)
340 % This function outputs three columns.
341 % (1) Excursion Time
342 % (2) Energy
343 % (3) Recombination Time
344 % (4) Trajectory number
345
346 % U_p = ((q^2)*(E0^2))/(4*(m_e)*(w_0^2)); % ponderomotive energy
347 timeColumn = DisplacementMatrix(:, 1);
348 collisionMatrix = RecombinationFinder(DisplacementMatrix);
349 collisionColumnTotal = length(collisionMatrix(1, :));
350
351 %continue with trying to make the first column initial times
352
353 % Find the maximum number of collisions, so that we can create our table.
354 % If this code is ever used to calculate a very high number of values, this
355 % bit could be improved speed-wise by just approximating cycle count
356 % instead of manually finding this out.
357
358 columnSums = zeros([1, collisionColumnTotal-1]);
359 for column = 2:1:collisionColumnTotal
360     columnSums(1, column-1) = sum(collisionMatrix(:, column));
361 end
362
363 ponderoColumnCount = max(columnSums);
364 ponderoRowCount = length(timeColumn);
365 BasicEnergyTable = zeros([ponderoRowCount, ponderoColumnCount]);
366
367 % Plot energy vs excursion time. To do this, we need to get the times
368 % corresponding to release and recapture. This is arguably all we need,
369 % because velocity only needs times, and constants to get velocity.
370
371 BasicEnergyTable(:, 1) = timeColumn;
372
373 for column = 2:1:collisionColumnTotal
374     % Find the indices of crossings within the displacement matrix
375     collisionIndices = find(collisionMatrix(:, column)==1);
376     % Find the index of the row where the trajectory begins
377     initialTimeRow = find(timeColumn==timeColumn(collisionIndices(1)));
378
379     % Go through collisionIndices and
380
381     for collision_column = 2:1:length(collisionIndices)
382         BasicEnergyTable(initialTimeRow, collision_column) = timeColumn(
                collisionIndices(collision_column)); %#ok<FNDSB>
383     end
384 end
385
386 findLastEjection = find(BasicEnergyTable(:, 1)>=lastEjectionTime, 1);
387 BasicEnergyTable = BasicEnergyTable(1:findLastEjection, :);
388 newRowCount = length(BasicEnergyTable(:, 1));
389
390 % Calculate all energy values
391
392 EnergyTable = BasicEnergyTable;
393 for column = 2:1:ponderoColumnCount
394     for row = 1:1:newRowCount
395         velValue = VelocityCalculator(q, E0, m_e, w_0, EnergyTable(row, column),
            EnergyTable(row, 1));
```

```
396         Energy = 0.5*m_e*velValue^2;
397         EnergyTable(row, column) = Energy;
398     end
399 end
400 EnergyTable = EnergyTable(:, 2:ponderoColumnCount);
401 SizeEn = size(EnergyTable);
402 EnergyRowTable = zeros([SizeEn(1)*SizeEn(2), 1]);
403 EnRIndex = 1;
404
405 % Calculate all excursion times
406
407 ExcursionTable = BasicEnergyTable;
408
409 recombinationTable = 0*BasicEnergyTable;
410
411 for column = 2:1:ponderoColumnCount
412     for row = 1:1:newRowCount
413         excursionValue = ExcursionTable(row, column) - ExcursionTable(row, 1);
414         ExcursionTable(row, column) = excursionValue;
415         recombinationTable(row, column) = BasicEnergyTable(row, column);
416     end
417 end
418 ExcursionTable = ExcursionTable(:, 2:ponderoColumnCount);
419 recombinationTable = recombinationTable(:, 2:ponderoColumnCount);
420 SizeEx = size(EnergyTable);
421 ExRowTable = zeros([SizeEx(1)*SizeEx(2), 1]);
422
423 ExcursionTable(ExcursionTable<=0) = 0;
424 BooleanTable = ExcursionTable==0;
425 EnergyTable(BooleanTable) = 0;
426 recombinationRowTable = ExRowTable;
427
428 for column = 1:1:length(EnergyTable(1, :))
429     for row = 1:1:newRowCount
430         if EnergyTable(row, column) ~= 0
431             EnergyRowTable(EnRIndex) = EnergyTable(row, column);
432             ExRowTable(EnRIndex) = ExcursionTable(row, column);
433             recombinationRowTable(EnRIndex) = recombinationTable(row, column);
434             EnRIndex = EnRIndex + 1;
435         else
436             EnergyRowTable(EnRIndex) = [];
437             ExRowTable(EnRIndex) = [];
438             recombinationRowTable(EnRIndex) = [];
439         end
440     end
441 end
442
443 CombinedTable = zeros([length(EnergyRowTable), 4]);
444 CombinedTable(:, 1) = ExRowTable;
445 CombinedTable(:, 2) = EnergyRowTable;
446 CombinedTable(:, 3) = recombinationRowTable;
447
448 CombinedTable = sortrows(CombinedTable, 1);
449
450 % With this now sorted by excursion time, the idea is now that we can go on
451 % and add a final column, which will be the number of the trajectory. This
452 % will be useful when only certain trajectories are desired.
453
454 % The first two trajectories are predictable. A maximum can be found
455 % between the beginning, and the end of the first time period.
```

```
456
457 TimePeriod = 1/(w_0/(2*pi));
458 beginningThirdTrajRow = find(CombinedTable(:, 1) > TimePeriod, 1);
459
460 firstAndSecondSnippet = CombinedTable(1:beginningThirdTrajRow, :);
461 maxEnergyIndex = find(CombinedTable(:, 2) == max(firstAndSecondSnippet(:, 2)));
462
463 CombinedTable(1:maxEnergyIndex-1, 4) = 1;
464 CombinedTable(maxEnergyIndex:beginningThirdTrajRow, 4) = 2;
465
466 TrajectoryNumber = 3;
467
468 for row = beginningThirdTrajRow+2:1:length(CombinedTable(:, 1))
469     latestDifference = CombinedTable(row, 2) - CombinedTable(row-1, 2);
470     previousDifference = CombinedTable(row-1, 2) - CombinedTable(row-2, 2);
471
472     % Scenario 1, function increasing
473     if latestDifference > 0 && previousDifference*latestDifference > 0
474         CombinedTable(row-2, 4) = TrajectoryNumber;
475     end
476     % Scenario 2, an energy peak
477     if latestDifference < 0 && previousDifference*latestDifference < 0
478         CombinedTable(row-2, 4) = TrajectoryNumber;
479         TrajectoryNumber = TrajectoryNumber + 1;
480     end
481     % Scenario 3, function decreasing
482     if latestDifference < 0 && previousDifference*latestDifference > 0
483         CombinedTable(row-2, 4) = TrajectoryNumber;
484     end
485     % Scenario 4, energy approaching 0
486     if latestDifference > 0 && previousDifference*latestDifference < 0
487         CombinedTable(row-2, 4) = TrajectoryNumber;
488         TrajectoryNumber = TrajectoryNumber + 1;
489     end
490     % Scenario 5, end of table. Assume same trajectory number.
491     if row == length(CombinedTable(:, 1))
492         CombinedTable(row, 4) = CombinedTable(row-2, 4);
493         CombinedTable(row-1, 4) = CombinedTable(row-2, 4);
494     end
495 end
496
497
498 BasicEnergyTable = CombinedTable;
499 end
500
501 function TrajectoryPlotter(DisplacementMatrix, CMap, q, E0, m_e, w_0)
502 % This function takes an input Matrix, and outputs a plot based upon that
503 % matrix.
504
505 % Choose a multiplier (10^9) will change the y axis from "meters" to
506 % "nanometers", etc. By default, "1" is meters.
507
508 DistanceMultiplier = 1;
509
510 timeColumn = DisplacementMatrix(:, 1);
511
512 collisionMatrix = RecombinationFinder(DisplacementMatrix);
513
514 hold on
515
```

```
516 % Plot the returning trajectories -- in order to use collisionMatrix
517 % function, must add time row back on
518
519 rowTotal = length(DisplacementMatrix(:, 1));
520 U_p = ((q^2)*(E0^2))/(4*(m_e)*(w_0^2)); % ponderomotive energy
521 maxEnergy = 3.18*U_p;
522 firstReturningTrajectoryColumn = 0;
523
524 for column = 2:1:(length(collisionMatrix(1, :))) %Incase something messes up, there
        was a "-1" next to the length?
525     collisionIndices = find(collisionMatrix(:, column)==1);
526     % First, plot all trajectories that diverge immediately
527     if length(collisionIndices) == 1
528         firstIndex = collisionIndices(1);
529         lastIndex = rowTotal;
530         DisplaceCut = DisplacementMatrix(firstIndex:lastIndex, column);
531         TimeCut = DisplacementMatrix(firstIndex:lastIndex, 1);
532         plot(TimeCut, DisplaceCut*DistanceMultiplier, 'Color', '[.8 .8 .8]');
533     end
534
535     % Next, plot all the reteurning trajectories
536     if length(collisionIndices) > 1
537
538         % Take a note of the first time this if statement is true, so that
539         % we can scale E0 based off of the maximum displacement later on
540
541         if firstReturningTrajectoryColumn == 0
542             firstReturningTrajectoryColumn = column;
543         end
544
545         % And now continue
546
547         t_i = timeColumn(collisionIndices(1));
548
549         for collisionIndex = 2:1:length(collisionIndices)
550             firstIndex = collisionIndices(collisionIndex-1);
551             lastIndex = collisionIndices(collisionIndex);
552             DisplaceCut = DisplacementMatrix(firstIndex:lastIndex, column);
553             TimeCut = timeColumn(firstIndex:lastIndex, 1);
554             t_r = timeColumn(lastIndex);
555             TrajVelocity = abs(VelocityCalculator(q, E0, m_e, w_0, t_r, t_i));
556             TrajEnergy = 0.5*m_e*TrajVelocity^2;
557             TrajColour = ColourMapUser(CMap, TrajEnergy, 0, maxEnergy);
558             plot(TimeCut, DisplaceCut*DistanceMultiplier, 'Color', TrajColour);
559
560             if collisionIndex == length(collisionIndices)
561                 firstIndex = collisionIndices(end);
562                 lastIndex = rowTotal;
563                 DisplaceCut = DisplacementMatrix(firstIndex:lastIndex, column);
564                 TimeCut = timeColumn(firstIndex:lastIndex, 1);
565                 plot(TimeCut, DisplaceCut*DistanceMultiplier, 'Color', '[.7 .7 .7]');
566             end
567
568         end
569     end
570 end
571
572 % First, we want to scale the electric field amplitude if it is to go on
573 % the same graph - scale it to half of the maximum displacement value, just
574 % so that it is more visually pleasing.
```

```
575
576 % You have to do "max of max" here because the max of an array returns the
577 % maximum of each column
578
579 maxDisplacement = max(max(DisplacementMatrix(:, firstReturningTrajectoryColumn:length
        (DisplacementMatrix(1, :)))));
580
581 yyaxis left
582
583 ylim([-1.2*maxDisplacement, 1.2*maxDisplacement]*DistanceMultiplier)
584 ylabel('Electron Displacement From Atom (m)', "Color", "k")
585
586 % Plot the electric field wave
587
588 % We choose the y axis bounds in a clever way so that the graph is forced
589 % to scale the electric field down to fit in the same graph, but retain its
590 % magnitude.
591
592 yyaxis right
593 set(gca,'YColor',[0 0 0]);
594 ylim([-1.2*2*E0, 1.2*2*E0])
595 ylabel('Electric Field Strength, E0 (V/m)', "Color", "k")
596 ElectricFieldLineColor = [34 139 34]/255;
597 plot(timeColumn, E0*sin(w_0.*timeColumn), 'LineWidth', 2.5, 'Color',
        ElectricFieldLineColor)
598 hold off
599
600 xlabel("Time (seconds)")
601
602 % Now just need to ensure that the rightside-y-axis is set up in such a way
603 % where this adjustment of the E0 amplitude makes sense.
604
605 grid on
606 end
607
608 function ElectricFieldValue_VoltsPerMetre = IntensityToElectricFieldCalculator(
        I_in_WattsperSquaredMetre, vacuumPermitivity_inFaradsperMetre,
        speedOfLight_inMetresPerSecond)
609     VoltsPerMetreE = sqrt((2*I_in_WattsperSquaredMetre)/(
            vacuumPermitivity_inFaradsperMetre*speedOfLight_inMetresPerSecond));
610     ElectricFieldValue_VoltsPerMetre = VoltsPerMetreE;
611 end
612
613 function RecombinationMatrix = RecombinationFinder(matrixTimeFirstCol)
614     % This function takes a displacement matrix and makes all values in the
615     % matrix 0, except the recombination points
616
617     M = matrixTimeFirstCol./abs(matrixTimeFirstCol);
618     trackerMatrix = 0*M;
619     trackerMatrix(:, 1) = matrixTimeFirstCol(:, 1);
620
621     trackerMatrix(isnan(trackerMatrix)) = 0;
622     ColumnNo = length(matrixTimeFirstCol(1, :));
623     RowNo = length(matrixTimeFirstCol(:, 1));
624     for column = 2:1:ColumnNo
625         for row = 2:1:RowNo
626             if isnan(M(row, column))
627                 trackerMatrix(row, column) = 0;
628             elseif M(row, column) ~= M(row-1, column)
629                 trackerMatrix(row, column) = 1;
```

```
630
631             elseif M(row, column) == M(row-1, column)
632                 trackerMatrix(row, column) = 0;
633             end
634         end
635     end
636     RecombinationMatrix = trackerMatrix;
637 end
638
639 function ColourMap = ColourTableCreator(IncrementCount, RGBLow, RGBHigh)
640     colourShades = IncrementCount;
641     lowestValueColour = RGBLow/256;
642     highestValueColour = RGBHigh/256;
643     colourRed = lowestValueColour(1) - highestValueColour(1);
644     colourGreen = lowestValueColour(2) - highestValueColour(2);
645     colourBlue = lowestValueColour(3) - highestValueColour(3);
646     colorInterval = [colourRed colourGreen colourBlue]/colourShades;
647     CreatedColourMap = zeros([colourShades length(lowestValueColour)]);
648     for row = 1:1:colourShades
649         CreatedColourMap(row, :) = lowestValueColour - row*colorInterval;
650     end
651     ColourMap = CreatedColourMap;
652 end
653
654 function ColourChoice = ColourMapUser(ColourMap, Value, MinValue, MaxValue)
655     Percentage = ((MinValue+Value)/(MinValue+MaxValue));
656     if Percentage > 1
657         Percentage = 1;
658     end
659     rowNumber = floor(Percentage*length(ColourMap(:, 1)));
660
661     if rowNumber < 1
662        rowNumber = 1;
663     end
664     ColourChoice = ColourMap(rowNumber, :);
665 end
666
667 function EnergyTableExtension = EnergyTableExtended(PonderoTable, h, w_0)
668     % This function takes an input PonderoTable and adds three columns to it
669     % -- approximated return time, quantum phase and classical phase. (5, 6 and 7)
670
671     % The input PonderoTable comes with four columns (1) Excursion Time,
672     % (2) Energy, (3) Recombination Time, (4) TrajectoryNumber
673
674     % First, make a small table of the frequency and t_r values, at the
675     % beginning of each new trajectory.
676
677     HighestTrajectoryNumber = max(PonderoTable(:, 4));
678     GroupDelayVsFreqTable = zeros([HighestTrajectoryNumber, 5]);
679
680     % This table will have the following columns (1) TrajectoryNumber (2)
681     % Frequency (3) t_r value (4) gradient value (5) intercept value
682
683     % Cycle through the PonderoTable and acquire information based upon
684     % trajectory.
685
686     for TrajectoryNumber = 1:1:HighestTrajectoryNumber
687         firstInstanceNewTrajectory = find(PonderoTable(:, 4)==TrajectoryNumber, 1);
688         % With the index now found, fill in the empty table
689         GroupDelayVsFreqTable(TrajectoryNumber, 1) = TrajectoryNumber;
```

```
690         GroupDelayVsFreqTable(TrajectoryNumber, 2) = PonderoTable(
                firstInstanceNewTrajectory, 2)/h;
691         GroupDelayVsFreqTable(TrajectoryNumber, 3) = PonderoTable(
                firstInstanceNewTrajectory, 3);
692     end
693
694     % Now, use these values to find the gradients of the trajectories if
695     % they are approximated by triangles
696
697     for Trajectory = 1:1:HighestTrajectoryNumber-1
698         GroupDelayVsFreqTable(Trajectory, 4) = (GroupDelayVsFreqTable(Trajectory+1,
                3)-GroupDelayVsFreqTable(Trajectory, 3))/(GroupDelayVsFreqTable(
                Trajectory+1, 2)-GroupDelayVsFreqTable(Trajectory, 2));
699     end
700
701     % Now, add the frequency = 0 intercepts, which should simply be a case
702     % of "stitching" the triangles together.
703
704     for Trajectory = 1:1:HighestTrajectoryNumber-1
705         GroupDelayVsFreqTable(Trajectory, 5) = GroupDelayVsFreqTable(Trajectory, 3) -
                GroupDelayVsFreqTable(Trajectory, 4)*GroupDelayVsFreqTable(Trajectory,
                2);
706     end
707
708     % We now have the required table, so now we want to re-shape the
709     % add another column onto the PonderoTable, which is "GroupDelayFit", a
710     % 5th column -- this will be the "group delay" according to the fit we
711     % just found.
712
713     % I also add another column here for the phase of the frequency, which
714     % will be found with an integral.
715
716     % First, initialise the new table we're going to use
717     PondRows = length(PonderoTable(:, 1));
718     PondColumns = length(PonderoTable(1, :));
719     modifiedPondTable = zeros(PondRows, PondColumns+3);
720     modifiedPondTable(1:PondRows, 1:PondColumns) = PonderoTable;
721
722     % Fill in the "fit" column
723
724     for row = 1:1:PondRows
725         modifiedPondTable(row, PondColumns+1) = (modifiedPondTable(row, 2)/h)*
                GroupDelayVsFreqTable(modifiedPondTable(row, 4), 4) +
                GroupDelayVsFreqTable(modifiedPondTable(row, 4), 5);
726     end
727
728     % Do a check to see if the final trajectory was complete -- If it
729     % wasn't, remove it.
730
731     if modifiedPondTable(end, PondColumns+1) == 0
732         firstInstanceOfLastTrajectory = find(modifiedPondTable(:, 4)==
                HighestTrajectoryNumber, 1);
733         modifiedPondTable(firstInstanceOfLastTrajectory:end, :) = [];
734     end
735
736     % Want to convert the Energy column into a frequency column for the
737     % next step.
738
739     modifiedPondTable(:, 2) = modifiedPondTable(:, 2)/h;
740
```

```
741     % Now, want to calculate the phase by taking the integral under the
742     % recombination time/frequency graph. Recalculate what the highest
743     % trajectory is -- it might now be different because of the deletion
744     % above.
745
746     Phi = 0;
747     for row = 2:1:length(modifiedPondTable(:, 1))
748         Phi = Phi + modifiedPondTable(row, 5)*(modifiedPondTable(row, 2)-
                modifiedPondTable(row-1, 2));
749         modifiedPondTable(row, 6) = Phi*2*pi;
750     end
751
752     % With the phase values now found, the second column is converted back
753     % to energy.
754
755     modifiedPondTable(:, 2) = modifiedPondTable(:, 2)*h;
756
757     % Add in classical phase
758
759     modifiedPondTable(:, 7) = (modifiedPondTable(:, 3).*(w_0))./(2*pi);
760
761     EnergyTableExtension = modifiedPondTable;
762 end
763
764 function DisplacementMatrix = DisplacementMatrixMaker(Time, EjectionTimes, q, E0, m_e
        , w_0)
765     % Create the raw matrix containing all values
766     % Ensure that we only use the required number of ejection times.
767
768     RowQty = length(Time)+1;
769     ColumnQty = length(EjectionTimes)+1;
770     Matrix = zeros([RowQty, ColumnQty]);
771     Matrix(1, 2:ColumnQty) = EjectionTimes;
772     Matrix(2:RowQty, 1) = Time;
773     for columnCounter = 2:1:ColumnQty
774         Matrix(2:RowQty, columnCounter) = DisplacementCalculator((Matrix(1,
                columnCounter)), Matrix(2:RowQty, 1), q, E0, m_e, w_0);
775     end
776
777     % Cleanse the data
778     % First, data before the first zero displacement should be wiped. (As
779     % this is the initial ionisation)
780
781     for columnCounter = 2:1:ColumnQty
782
783         % We scan across the columns, one column at a time. We cut out the
784         % column of interest, taking a cut that ignores the initial time.
785
786         columnOfInterest = Matrix(2:RowQty, columnCounter);
787
788         for row = 2:1:RowQty
789             errorRow = -1;
790             if abs(row - columnCounter) >= 10
791                 % This extra if statement arose from an interesting bug
792                 % that seems to happen. In some instances, the value
793                 % calculated by MATLAB is outright incorrect -- It does not
794                 % intercept when it should. This is to prevent it
795                 % "skipping" the intercept, so that this can be fixed
796                 % "manually" later on
797                 errorRow = row;
```

```
798                 continue
799             end
800             if columnOfInterest(row, :)*columnOfInterest(row-1, :) <= 0
801                 % If there is a sign change, note it and break out of
802                 % the row loop
803                 zeroIndex = row+1;
804                 break
805             else
806                 % If there is not a sign change, continue to next row
807                 continue
808             end
809         end
810
811         if errorRow == row
812             Matrix(2:zeroIndex+1, columnCounter) = 0;
813         else
814             Matrix(2:zeroIndex, columnCounter) = 0;
815         end
816
817
818     end
819
820     % Removing the first row, because initial time now doesn't need its own
821     % row, but can be seen as the first "0" time from the first column.
822
823     Matrix = Matrix(2:RowQty, :);
824
825     % Output matrix
826     DisplacementMatrix = Matrix;
827 end
828
829 function DisplacementValue = DisplacementCalculator(t_i, Time, q, E0, m_e, w_0)
830     DisplacementValue = ((q*E0)/(m_e*w_0^2))*(sin(w_0.*Time) - sin(w_0*t_i) - w_0*(
        Time - t_i)*cos(w_0*t_i));
831 end
832
833 function VelocityValue = VelocityCalculator(q, E0, m_e, w_0, returnTime, initialTime)
834     VelocityValue = abs(((q*E0)/(m_e*w_0))*((cos(w_0*returnTime)) - cos(w_0*
        initialTime)));
835 end
```

References

[1] S. Ghimire and D. A. Reis, "High-harmonic generation from solids," *Nature physics*, vol. 15, no. 1, pp. 10–16, 2019.

[2] T. Brabec and F. Krausz, "Intense few-cycle laser fields: Frontiers of nonlinear optics," *Rev. Mod. Phys.*, vol. 72, pp. 545–591, Apr 2000.

[3] N. Ibrakovic, *Control of Coherent Extreme Ultraviolet Light and Light Sources*. PhD thesis, Lund University, 2019.

[4] I. Hendriks, "Phase control with spatial-light modulators towards application for opto-optical modulation," 2020.

[5] M. Lewenstein, P. Balcou, M. Y. Ivanov, A. L'huillier, and P. B. Corkum, "Theory of high-harmonic generation by low-frequency laser fields," *Physical Review A*, vol. 49, no. 3, p. 2117, 1994.

[6] K. L. Ishikawa, "High-harmonic generation," *Advances in Solid State Lasers Development and Applications*, pp. 439–465, 2010.

[7] J. M. Dahlström, *Strong Field Approximation for High Order Harmonic Generation with $\omega/2\omega$ Laser Fields*. PhD thesis, Lund University, 2007.

[8] S. Ek, "Evaluation of the wavefronts and spatial structures of ultrashort pulses passing through a micro-channel plate," 2019.

[9] K. Varjú‖, Y. Mairesse, B. Carré, M. B. Gaarde, P. Johnsson, S. Kazamias, R. López-Martens, J. Mauritsson, K. Schafer, P. Balcou, *et al.*, "Frequency chirp of harmonic and attosecond pulses," *Journal of Modern Optics*, vol. 52, no. 2-3, pp. 379–394, 2005.

[10] C. Guo, *A high repetition rate attosecond light source based on optical parametric amplification*. Lund University, 2018.

[11] C. E. Shannon, "Communication in the presence of noise," *Proceedings of the IRE*, vol. 37, no. 1, pp. 10–21, 1949.

[12] H. Nyquist, "Certain topics in telegraph transmission theory," *Transactions of the American Institute of Electrical Engineers*, vol. 47, no. 2, pp. 617–644, 1928.

[13] M. Jonathan and T. Martin, "Digital signal processing: Mathematical and computational methods, software development and applications," 2006.

[14] G. Fan, K. Legare, V. Cardin, X. Xie, E. Kaksis, G. Andriukaitis, A. Pugzlys, B. Schmidt, J. Wolf, M. Hehn, *et al.*, "Time-resolving magnetic scattering on rare-earth ferrimagnets with a bright soft-x-ray high-harmonic source," *arXiv preprint arXiv:1910.14263*, 2019.